目录

[#数论 2](#_Toc22916453)

[### 快速乘 2](#_Toc22916454)

[### 快速幂 2](#_Toc22916455)

[### 逆元 2](#_Toc22916456)

[### 计算n!中因子p的指数 3](#_Toc22916457)

[### 用筛法生成素数 3](#_Toc22916458)

[### 线性筛 筛欧拉 3](#_Toc22916459)

[###分解质因数求欧拉函数 3](#_Toc22916460)

[### 埃式筛求欧拉 3](#_Toc22916461)

[### 线性筛约数个数 4](#_Toc22916462)

[### 线性筛约数和 4](#_Toc22916463)

[### 递归欧拉降幂 5](#_Toc22916464)

[### MillerRobbin判素数 5](#_Toc22916465)

[### 区间筛质数 6](#_Toc22916466)

[### 约数个数 6](#_Toc22916467)

[### 区间筛约数个数 6](#_Toc22916468)

[### 数值分块 7](#_Toc22916469)

[### 高斯消元 7](#_Toc22916470)

[### exgcd 8](#_Toc22916471)

[### 中国剩余定理 8](#_Toc22916472)

[### BSGS 8](#_Toc22916473)

[### 组合数 9](#_Toc22916474)

[### 卡特兰数 10](#_Toc22916475)

[### 莫比乌斯反演 11](#_Toc22916476)

[#计算几何 15](#_Toc22916477)

[### 基本操作（点、线、圆） 15](#_Toc22916478)

[### 极角排序 18](#_Toc22916479)

[### 二维凸包 19](#_Toc22916480)

[### 最大空凸包 20](#_Toc22916481)

[### 三维凸包 21](#_Toc22916482)

[### 半平面交 24](#_Toc22916483)

[### 凸多边形最大内切圆 26](#_Toc22916484)

[###最小覆盖球 27](#_Toc22916485)

[###最小圆覆盖 30](#_Toc22916486)

[### 最近平面点对 31](#_Toc22916487)

[#常用操作 31](#_Toc22916488)

[### 二维差分 31](#_Toc22916489)

[### 二维前缀和 32](#_Toc22916490)

[### 多维前缀和 32](#_Toc22916491)

[### 离散化 32](#_Toc22916492)

[### 高精度加法 32](#_Toc22916493)

[### 高精度减法 32](#_Toc22916494)

[### 高精度乘法 32](#_Toc22916495)

[### 高精度除法 32](#_Toc22916496)

[### 高精度比较 33](#_Toc22916497)

[### 数组模拟邻接表 33](#_Toc22916498)

[### 快读快输 33](#_Toc22916499)

[###二分查找 33](#_Toc22916500)

[### 矩阵快速幂 33](#_Toc22916501)

[#背包问题 34](#_Toc22916502)

[### 01背包问题 34](#_Toc22916503)

[### 完全背包问题 34](#_Toc22916504)

[###多重背包问题 I 35](#_Toc22916505)

[###多重背包问题 II 35](#_Toc22916506)

[###多重背包问题 III 35](#_Toc22916507)

[###混合背包问题 36](#_Toc22916508)

[###二维费用的背包问题 36](#_Toc22916509)

[###分组背包问题 37](#_Toc22916510)

[###有依赖的背包问题 37](#_Toc22916511)

[###背包问题求方案数 38](#_Toc22916512)

[###背包问题求具体方案 38](#_Toc22916513)

[#Trie树 39](#_Toc22916514)

[### 前缀统计 39](#_Toc22916515)

[###最大异或对 39](#_Toc22916516)

[###最大异或值路径 40](#_Toc22916517)

[# 字符串 41](#_Toc22916518)

[### KMP 41](#_Toc22916519)

[### 扩展KMP 42](#_Toc22916520)

[### Manacher 42](#_Toc22916521)

[#其它 43](#_Toc22916522)

[### CDQ分治二维（逆序对） 43](#_Toc22916523)

[### CDQ分治三维（陌上花开） 43](#_Toc22916524)

[### CDQ分治四维 45](#_Toc22916525)

[### 树状数组 46](#_Toc22916526)

[### 莫队 47](#_Toc22916527)

[###线段树 49](#_Toc22916528)

[### 最小生成树 51](#_Toc22916529)

[### floyd 52](#_Toc22916530)

[### 单调队列 52](#_Toc22916531)

[### 第K小完全图 53](#_Toc22916532)

# #数论

### 快速乘

ll mul(ll a, ll b, ll m) {

ll ret = 0;

while (b) {

if (b & 1) {

ret += a;

if (ret >= m) ret -= m;

}

a += a;

if (a >= m) a -= m;

b >>= 1;

}

return ret;

}

**O(1)**

**ll mul(ll u, ll v, ll p) {**

**return (u \* v - ll((long double) u \* v / p) \* p + p) % p;**

**}**

**ll mul(ll u, ll v, ll p) { // 卡常**

**ll t = u \* v - ll((long double) u \* v / p) \* p;**

**return t < 0 ? t + p : t;**

**}**

**### 快速幂**

//如果模数是素数，则可在函数体内加上n %= mod - 1;（费马小定理）。

ll ksm(ll x, ll n, ll mod) {

ll ret = mod != 1;

for (x %= mod; n; n >>= 1, x = x \* x % mod)

if (n & 1) ret = ret \* x % mod;

return ret;

}

//防爆 ll

//前置模板：快速乘

ll ksm(ll x, ll n, ll mod) {

ll ret = mod != 1;

for (x %= mod; n; n >>= 1, x = mul(x, x, mod))

if (n & 1) ret = mul(ret, x, mod);

return ret;

}

### 逆元

1.费马小定理（a^(mod-2)）

2.利用扩展欧几里得

const ll mod = 9973;//改变

ll exgcd(ll a, ll b, ll &x, ll &y)

{

if(!b)

{

x = 1; y = 0;

return a;

}

int d = exgcd(b, a % b, y, x);

y -= (a/b) \* x;

return d;

}

ll inev(ll n) // m/n = 1 (%mod)求n的逆元

{

ll a=n,b=mod,c=1,x,y;

ll cur=exgcd(a,b,x,y);

if(c%cur)

return -1;

x\*=c/cur;

b/=cur;

if(b<0)

b=-b;

ll ans=(x%b+b)%b;

return ans;

}

3.O(n)递推算法

inv[1] = 1;

for(int i = 2; i < p; ++ i)

inv[i] = (p - p / i) \* inv[p % i] % p;

4.递推求阶乘逆元

void init() {

fact[0] = 1;

for (int i = 1; i < maxn; i++) {

fact[i] = fact[i - 1] \* i %mod;

}

inv[maxn - 1] = ksm(fact[maxn - 1], mod - 2);

for (int i = maxn - 2; i >= 0; i--) {

inv[i] = inv[i + 1] \* (i + 1) %mod;

}

}

### 计算n!中因子p的指数

ll cal(ll n,ll p)

{

ll t = n;

ll cnt = 0;

while (t)

{

cnt += t / p;

t /= p;

}

return cnt;

}

### 用筛法生成素数

const int N = 100005;

bool prime[N];

int p[N],tot;

void init()

{

for(int i=2;i<N;i++)

prime[i]=true;

for(int i=2;i<N;i++)

{

if(prime[i])

p[tot++]=i;

for(int j=0;j<tot&&p[j]\*i<=N;j++)

{

prime[p[j]\*i]=false;

if(i%p[j]==0)

break;

}

}

}

### 线性筛 筛欧拉

bool st[N];

int prime[N],tot;

ll phi[N];

void init()

{

phi[1]=1;

for(int i=2;i<N;i++)

st[i]=1;

for(int i=2;i<N;i++)

{

if(st[i])

{

prime[tot++]=i;

phi[i]=i-1;

}

for(int j=0;j<tot&&prime[j]\*i<N;j++)

{

st[prime[j]\*i]=0;

if(i%prime[j]==0)

{

phi[i\*prime[j]]=phi[i]\*prime[j];

break;

}

else

phi[i\*prime[j]]=phi[i]\*(prime[j]-1);

}

}

}

###分解质因数求欧拉函数

int phi(int n)

{

int ans=n;

for(int i=2;i\*i<=n;i++)

{

if(n%i==0)

ans=ans/i\*(i-1);

while(n%i==0)

n/=i;

}

if(n>1)

ans=ans/n\*(n-1);

return ans;

}

### 埃式筛求欧拉

int phi[MAXN];

void euler(int n)

{

for(int i=1;i<=n;i++)

phi[i]=i;

for(int i=2;i<=n;i++)

{

if(phi[i]==i)//这代表i是质数

{

for(int j=i;j<=n;j+=i)//把i的倍数更新掉

phi[j]=phi[j]/i\*(i-1);

}

}

}

### 线性筛约数个数

const int N=1e5+5;

bool mark[N];

int prim[N],d[N],num[N];

//d(i) 表示 i 的约数个数

//num[i] 表示 i 的最小素因子的个数

//prim[i] 表示 第 i 个素数

int cnt;

void initial()

{

cnt=0;

d[1]=1;

for (int i=2 ; i<N ; ++i)

{

if (!mark[i])

{

prim[cnt++]=i;

num[i]=1;

d[i]=2;

}

for (int j=0 ; j<cnt && i\*prim[j]<N ; ++j)

{

mark[i\*prim[j]]=1;

if (!(i%prim[j]))

{

num[i\*prim[j]]=num[i]+1;

d[i\*prim[j]]=d[i]/(num[i]+1)\*(num[i\*prim[j]]+1);

break;

}

d[i\*prim[j]]=d[i]\*d[prim[j]];

num[i\*prim[j]]=1;

}

}

}

### 线性筛约数和

https://img-blog.csdn.net/20170828171139194

const int N=1e6+5;

bool mark[N];

int prim[N];

ll sd[N],sp[N];

//sd(i) 表示 i 的约数和

//sp[i] 表示 i 的最小素因子的等比数列的和

//prim[i] 表示第 i 个素数

int cnt;

void initial()

{

cnt=0;

sd[1]=1;

for (int i=2 ; i<N ; ++i)

{

if (!mark[i])

{

prim[cnt++]=i;

sd[i]=i+1;

sp[i]=i+1;

}

for (int j=0 ; j<cnt && i\*prim[j]<N ; ++j)

{

mark[i\*prim[j]]=1;

if (!(i%prim[j]))

{

sp[i\*prim[j]]=sp[i]\*prim[j]+1;

sd[i\*prim[j]]=sd[i]/sp[i]\*sp[i\*prim[j]];

break;

}

sd[i\*prim[j]]=sd[i]\*sd[prim[j]];

sp[i\*prim[j]]=1+prim[j];

}

}

}

### 递归欧拉降幂

const int MAXN = 1e4+10;

int phi[MAXN],flag[MAXN];

vector<int>p;//存储质数

void euler(int n)//筛欧拉

{

phi[1]=1;

for(int i=2;i<=n;i++)

{

if(flag[i]==0)

{

p.push\_back(i);

phi[i]=i-1;

}

for(int j=0;j<p.size()&&i\*p[j]<=n;j++)

{

flag[i\*p[j]]=1;

if(i%p[j]==0)

{

phi[i\*p[j]]=phi[i]\*p[j];

break;

}

phi[i\*p[j]]=phi[i]\*phi[p[j]];

}

}

}

int M,n,a[MAXN];//a数组记录幂次

ll ksm(ll x, ll k,ll mod)

{

ll res = 1;

ll m = x;

while (k)

{

if (k & 1)

res = res \* m % mod;

m = m \* m % mod;

k >>= 1;

}

return res;

}

ll f(int pos,ll mod)

{

if(pos==n)

return a[pos]%M;

return ksm(a[pos],f(pos+1,phi[mod])+phi[mod],mod);

}

//ll f(ll m,ll mod)//n^n^n...^n 一共m个

//{

// if(mod == 1) return 0;

// if(m==0) return 1;

// ll t = phi[mod];

// ll cc=f(m-1,t);

// if(cc<t&&cc) return ksm(a[m],cc,mod);

// else return ksm(a[m],cc+t,mod);

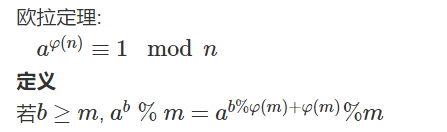
//}

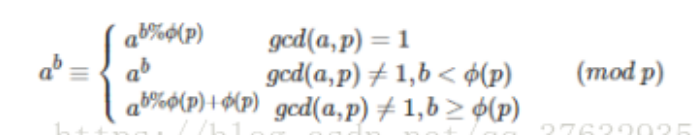
int main()

{

ll res=f(1,mod);//返回答案

}



****

### MillerRobbin判素数

bool check(ll a, ll n)

{

ll m = n - 1, x, y;

int i, j = 0;

while (!(m & 1)) m >>= 1, j++;

x = ksm(a, m, n);

for (i = 1; i <= j; x = y, i++)

{

y = ksm(x, 2, n);

if ((y == 1) && (x != 1) && (x != n - 1)) return 1;

}

return y != 1;

}

bool miller\_rabin(int times, ll n)

{

ll a;

if (n == 1) return 0;

if (n == 2) return 1;

if (!(n & 1)) return 0;

while (times--)

{

if (check(rand() % (n - 1) + 1, n))

return 0;

}

return 1;

}

### 区间筛质数

void interval\_sieve()//[L,R]间的质数

{

for(int i=0;i<MAXN;i++)

v[i]=1;

if(L==1)//特判1的时候

v[0]=0;

for(ll i=0;i<prime.size();i++)

{

for(ll j=(L-1)/prime[i]+1;j<=R/prime[i];j++)

{

if(j>1)

v[prime[i]\*j-L]=0;

}

}

}

### 约数个数

**Problem 1**

求A^B的约数之和

int a,b;

vector<ll>q;

ll ksm(ll a,ll p)

{

ll res=1;

a=a%mod;

while(p)

{

if(p&1)

res=res\*a%mod;

a=a\*a%mod;

p>>=1;

}

return res;

}

int sum(int a,int p)

{

if(p==0)

return 1;

if(p%2==0)//奇数偶数分类讨论

return (1+(a%mod)\*sum(a,p-1))%mod;

else

return (1+ksm(a,p/2+1))\*sum(a,p/2)%mod;

}

int main()

{

cin>>a>>b;

if(a==0)

{

cout<<0<<endl;

return 0;

}

int cou=1;

for(int i=2;i<=a;i++)

{

int ans=0;

while(a%i==0)

{

a/=i;

ans++;

}

ans\*=b;

if(ans>0)

cou=cou\*sum(i,ans)%mod;

}

cout<<cou<<endl;

}

### 区间筛约数个数

ll interval\_divisior\_count()//[L,R]内约数的个数

{

for(ll i=0;i<=R-L;i++)

sum[i]=1,res[i]=i+L;

for(int i=0;i<prime.size();i++)

{

ll temp=((L-1)/prime[i]+1)\*prime[i];

for(ll j=temp;j<=R;j+=prime[i])

{

ll cnt=0;

while(res[j-L]%prime[i]==0)

{

cnt++;

res[j-L]/=prime[i];

}

sum[j-L]=sum[j-L]\*((cnt%mod)\*(k%mod)%mod+1)%mod;

//这里的k是指 题目中求了i的k次的约数个数

}

}

ll ans=0;

for(int i=0;i<=R-L;i++)

{

if(res[i]==1)

ans=(ans+sum[i])%mod;

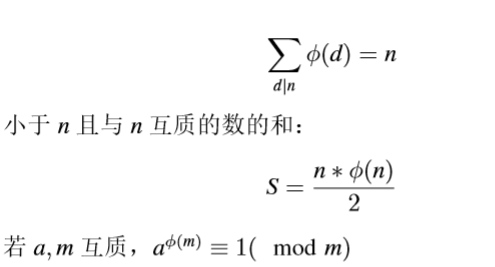
else

ans=(ans+(sum[i]%mod)\*((k+1)%mod)%mod)%mod;

}

return ans;

}



### 数值分块

ll get\_sum\_of\_remainder(ll n,ll k)

{//计算k%1+k%2+k%3+…+k%n

ll ans=n\*k;

if(n>k)

n=k;

for(ll l=1,r;l<=n;l=r+1)

{

r=min(k/(k/l),n);

ans-=(k/l)\*(l+r)\*(r-l+1)/2;

}

return ans;

}

### 高斯消元

int lcm(int a,int b)

{

return a/\_\_gcd(a,b)\*b;

}

void Gauss(int equ,int var)//行、列

{//答案a[i][var]为解

int i,j,k;

int max\_r,col=0;

int LCM,ta,tb;

for(k=0;k<equ&&col<var;k++,col++)// 枚举当前处理的行

// 找到该col列元素绝对值最大的那行与第k行交换.(为了在除法时减小误差)

{

max\_r=k;

for(i=k+1;i<equ;i++)

if(abs(a[i][col])>abs(a[max\_r][col]))

max\_r=i;

if(max\_r!=k)// 与第k行交换.

{

for(j=k;j<var+1;j++)

swap(a[k][j],a[max\_r][j]);

}

if(a[k][col]==0)// 说明该col列第k行以下全是0了，则处理当前行的下一列.

{

k--;

continue;

}

for(i=k+1;i<equ;i++)// 枚举要删去的行

{

if(a[i][col]!=0)

{

LCM=lcm(abs(a[i][col]),abs(a[k][col]));

ta=LCM/(abs(a[i][col]));

tb=LCM/(abs(a[k][col]));

if(a[i][col]\*a[k][col]<0)//异号的情况是相加

tb=-tb;

for(j=col;j<var+1;j++)

a[i][j]=(a[i][j]\*ta-a[k][j]\*tb)%mod;

}

}

}

return;

}

### exgcd

//a\*x+b\*y=c;求最小整数x

ll A,B,m,n,l;

ll exgcd(ll a,ll b,ll &x,ll &y){

if(b == 0){

x = 1;

y = 0;

return a;

}

ll gcd = exgcd(b,a%b,y,x);

y -= a / b \* x;

return gcd;

}

int main()

{

cin>>A>>B>>m>>n>>l;

ll a=m-n,b=l,c=B-A,x,y;

ll cur=exgcd(a,b,x,y);

// cout<<c<<" "<<cur<<endl;

if(c%cur!=0)

cout<<"Impossible\n";

else

{

x\*=c/cur;

b/=cur;

if(b<0)

b=-b;

// cout<<x<<" "<<b<<endl;

ll ans=(x%b+b)%b;

cout<<ans<<endl;

}

}

### 中国剩余定理

const int N = 1e4+5;

ll n,m,k,a[N],b[N];

ll exgcd(ll a, ll b, ll &x, ll &y)

{

if (!b)

{

x = 1;

y = 0;

return a;

}

int d = exgcd(b, a % b, y, x);

y -= (a/b) \* x;

return d;

}

ll china(ll \*m,ll \*a) //方程x%m=a;

{

ll lcm=1,X=m[1],Y=a[1];

for(int i=1;i<=k;i++)

lcm=lcm/\_\_gcd(lcm,m[i])\*m[i];

for(int i=2;i<=k;i++)

{

ll A=X,B=m[i],d,x,y,c=a[i]-Y;

d=exgcd(A,B,x,y);

if(c%d)return -1;

ll mod=m[i]/d;

ll K=((x\*c/d)%mod+mod)%mod;

Y=X\*K+Y;

X=X\*m[i]/d;

}

if(Y==0)

return lcm;

return Y;

}

### BSGS

ll bsgs(ll a,ll b,ll p)//a^x ≡b (mod p) (a,p互质,求最小的正整数x)

{

map<ll,ll>H;

H.clear();

b%=p;

ll t=(ll)sqrt(p)+1;

for(ll i=0;i<t;i++)

{

ll val=(ll)b\*ksm(a,i,p)%p;

H[val]=i;

}

a=ksm(a,t,p);

if(!a)

{

if(b==0)

return 1;//返回最小的正整数1

else

return -1;

}

for(ll i=0;i<=t;i++)

{

ll val=ksm(a,i,p);

ll j;

if(H.find(val)==H.end())

j=-1;

else

j=H[val];

if(j>=0&&i\*t>=j)

return i\*t-j;

}

return -1;

}

### 组合数

**方法1：**

首先预处理出所有阶乘取模的余数fact[N]，以及所有阶乘取模的逆元infact[N]

如果取模的数是质数，可以用费马小定理求逆元

int qmi(int a, int k, int p) // 快速幂模板

{

int res = 1;

while (k)

{

if (k & 1) res = (LL)res \* a % p;

a = (LL)a \* a % p;

k >>= 1;

}

return res;

}

// 预处理阶乘的余数和阶乘逆元的余数

fact[0] = infact[0] = 1;

for (int i = 1; i < N; i ++ )

{

fact[i] = (LL)fact[i - 1] \* i % mod;

infact[i] = (LL)infact[i - 1] \* qmi(i, mod - 2, mod) % mod;

}

**方法2：**Lucas定理

若p是质数，则对于任意整数 1 <= m <= n，有：

C(n, m) = C(n % p, m % p) \* C(n / p, m / p) (mod p)

int qmi(int a, int k) // 快速幂模板

{

int res = 1;

while (k)

{

if (k & 1) res = (LL)res \* a % p;

a = (LL)a \* a % p;

k >>= 1;

}

return res;

}

int C(int a, int b) // 通过定理求组合数C(a, b)

{

int res = 1;

for (int i = 1, j = a; i <= b; i ++, j -- )

{

res = (LL)res \* j % p;

res = (LL)res \* qmi(i, p - 2) % p;

}

return res;

}

int lucas(LL a, LL b)

{

if (a < p && b < p) return C(a, b);

return (LL)C(a % p, b % p) \* lucas(a / p, b / p) % p;

}

**方法3：**分解质因数法求组合数（求真实值）

int primes[N], cnt; // 存储所有质数

int sum[N]; // 存储每个质数的次数

bool st[N]; // 存储每个数是否已被筛掉

void get\_primes(int n) // 线性筛法求素数

{

for (int i = 2; i <= n; i ++ )

{

if (!st[i]) primes[cnt ++ ] = i;

for (int j = 0; primes[j] <= n / i; j ++ )

{

st[primes[j] \* i] = true;

if (i % primes[j] == 0) break;

}

}

}

int get(int n, int p) // 求n！中的次数

{

int res = 0;

while (n)

{

res += n / p;

n /= p;

}

return res;

}

vector<int> mul(vector<int> a, int b) // 高精度乘低精度模板

{

vector<int> c;

int t = 0;

for (int i = 0; i < a.size(); i ++ )

{

t += a[i] \* b;

c.push\_back(t % 10);

t /= 10;

}

while (t)

{

c.push\_back(t % 10);

t /= 10;

}

return c;

}

get\_primes(a); // 预处理范围内的所有质数

for (int i = 0; i < cnt; i ++ ) // 求每个质因数的次数

{

int p = primes[i];

sum[i] = get(a, p) - get(b, p) - get(a - b, p);

}

vector<int> res;

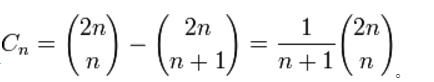
res.push\_back(1);

for (int i = 0; i < cnt; i ++ ) // 用高精度乘法将所有质因子相乘

for (int j = 0; j < sum[i]; j ++ )

res = mul(res, primes[i]);

### 卡特兰数



https://images2018.cnblogs.com/blog/1384700/201806/1384700-20180605202529225-1568538317.png　　(其中n>=2)

https://images2018.cnblogs.com/blog/1384700/201806/1384700-20180605202617818-56686056.png

上式解为　https://images2018.cnblogs.com/blog/1384700/201806/1384700-20180605202718459-1444705381.png　　(n=1,2,3,...)

//数据1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 742900, 2674440, 9694845, 35357670, 129644790, 477638700, 1767263190, 6564120420, 24466267020, 91482563640, 343059613650, 1289904147324,

**应用**：

1. 括号化问题。矩阵链乘： P=A1×A2×A3×……×An，依据乘法结合律，不改变其顺序，只用括号表示成对的乘积，试问有几种括号化的方案
2. 将多边行划分为三角形问题。将一个凸多边形区域分成三角形区域(划分线不交叉)的方法数
3. 在圆上选择2n个点,将这些点成对连接起来使得所得到的n条线段不相交的方法数
4. 出栈次序问题。一个栈(无穷大)的进栈序列为1、2、3、...、n，有多少个不同的出栈序列
5. 一位大城市的律师在他住所以北n个街区和以东n个街区处工作，每天她走2n个街区去上班。如果他从不穿越（但可以碰到）从家到办公室的对角线，那么有多少条可能的道路
6. 给定N个节点，能构成多少种形状不同的二叉树(1,1,4,..)
7. 求01串的个数：n个0与n个1构成的序列方案数，使得任何一个前缀0的个数不少于1的个数

//计算真实值

#include<bits/stdc++.h>

using namespace std;

typedef long long ll;

const int N = 120005;

bool prime[N];

int p[N],tot,num[N],n;

vector<ll>res;

void init()

{

for(int i=2;i<N;i++)

prime[i]=true;

for(int i=2;i<N;i++)

{

if(prime[i])

p[tot++]=i;

for(int j=0;j<tot&&p[j]\*i<=N;j++)

{

prime[p[j]\*i]=false;

if(i%p[j]==0)

break;

}

}

}

int cou(int n,int p)

{

int ans=0;

while(n)

{

ans+=n/p;

n=n/p;

}

return ans;

}

void mul(int n)

{

ll temp=0;

for(int i=0;i<res.size();i++)

{

ll cur=temp+res[i]\*n;

res[i]=cur%1000000000;

temp=cur/1000000000;

}

while(temp)

{

res.push\_back(temp%1000000000);

temp/=1000000000;

}

}

int main()

{

init();

cin>>n;

for(int i=0;i<tot;i++)

num[p[i]]+=cou(2\*n,p[i])-cou(n,p[i])-cou(n+1,p[i]);

res.push\_back(1);

for(int i=0;i<tot;i++)

{

for(int j=0;j<num[p[i]];j++)

mul(p[i]);

}

printf("%lld",res.back());

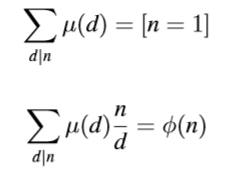
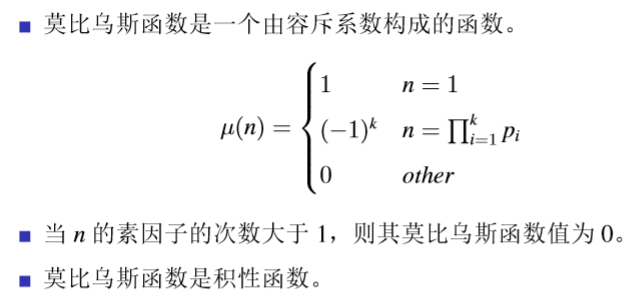
for(int i=res.size()-2;i>=0;i--)

printf("%09lld",res[i]);

}

### 莫比乌斯反演

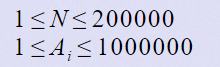






**Problem 1**

任选两个数的lcm之和

  
F(i):gcd为i的倍数的数对的乘积和;

f(x):gcd为i的数对的乘积和



#include<bits/stdc++.h>

using namespace std;

typedef long long ll;

const int N = 1e6+10;

const ll mod = 998244353;

int prime[N],mu[N],tot;

bool vis[N];

ll ksm(ll a,ll p)

{

ll res=1;

while(p)

{

if(p&1)

res=res\*a%mod;

a=a\*a%mod;

p>>=1;

}

return res;

}

void sieve()

{

memset(vis,1,sizeof vis);

vis[0]=vis[1]=0;

mu[1]=1;

for(int i=2;i<N;i++)

{

if(vis[i])

{

prime[tot++]=i;

mu[i]=-1;

}

for(int j=0;j<tot&&1ll\*i\*prime[j]<N;j++)

{

int x=prime[j];

vis[i\*x]=0;

if(i%x==0)

{

mu[i\*x]=0;

break;

}

else

mu[i\*x]=-mu[i];

}

}

}

ll n,a[N],f[N],cnt[N],ma;

int main()

{

sieve();

scanf("%lld",&n);

for(int i=1;i<=n;i++)

{

scanf("%lld",&a[i]);

cnt[a[i]]++;

ma=max(ma,a[i]);

}

for(int i=1;i<=ma;i++)

{

ll sum=0;

for(int j=i;j<=ma;j+=i)

{

f[i]=(f[i]+cnt[j]\*(cnt[j]-1)/2\*j%mod\*j%mod+sum\*j\*cnt[j]%mod)%mod;

sum=(sum+j\*cnt[j]%mod)%mod;

}

}

ll ans=0;

for(int i=1;i<=ma;i++)

{

ll now=0;

for(int j=i;j<=ma;j+=i)

now=(f[j]\*mu[j/i]%mod+now)%mod;

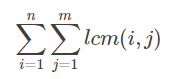
ans=(ans+now\*ksm(i,mod-2))%mod;

}

printf("%lld\n",ans);

}

**Problem 2**



N, M ≤ 10^7

**Solution 1:**

**//O(n)**

#include <bits/stdc++.h>

#define ll long long

#define N 10000010

#define mod 20101009

using namespace std;

int n,m,prime[N],tot=0;

ll mu[N];

bool notprime[N];

void Mobius()

{

memset(notprime,0,sizeof(notprime));

mu[1]=1;

notprime[1]=1;

for(int i=2; i<=n; ++i)

{

if(!notprime[i])

{

prime[++tot]=i;

mu[i]=-1;

}

for(int j=1; prime[j]\*i<=n; ++j)

{

notprime[prime[j]\*i]=1;

if(i%prime[j]==0)

{

mu[prime[j]\*i]=0;

break;

}

mu[prime[j]\*i]=-mu[i];

}

}

for(int i=1; i<=n; ++i)

mu[i]=((ll)mu[i]\*i%mod\*i%mod+mu[i-1])%mod;

}

inline ll sum(ll x,ll y)

{

return x\*(x+1)/2%mod\*(y\*(y+1)/2%mod)%mod;

}

ll f(ll x,ll y)

{

if(x>y) swap(x,y);

ll re=0;

ll last;

for(ll i=1; i<=x; i=last+1)

{

last=min(x/(x/i),y/(y/i));

re=(re+(mu[last]-mu[i-1]+mod)\*sum(x/i,y/i)%mod)%mod;

//可能会减出负的。要注意。

}

return re;

}

int main()

{

scanf("%d%d",&n,&m);

if(n>m) swap(n,m);

Mobius();

ll last;

ll ans=0;

for(ll i=1; i<=n; i=last+1)

{

last=min(n/(n/i),m/(m/i));

ans=(ans+(last+i)\*(last-i+1)/2%mod\*f(n/i,m/i))%mod;

}

printf("%lld\n",ans);

return 0;

}

**Solution 2**

**//O(√n)**

#include<bits/stdc++.h>

using namespace std;

const int N = 1e7;

typedef long long ll;

const ll mod = 1e8 + 9;

int read()

{

char ch = getchar(); int x = 0, f = 1;

while(ch < '0' || ch > '9') {if(ch == '-') f = -1; ch = getchar();}

while(ch >= '0' && ch <= '9') {x = x \* 10 + ch - '0'; ch = getchar();}

return x \* f;

}

int top, n, m, pri[N + 10];

ll g[N + 10];

bool vis[N + 10];

int get\_miu() {

g[1] = 1;

for(int i = 2;i <= N; ++i) {

if(!vis[i]) {

g[i] = 1LL \* i \* (1 - i) % mod;

pri[++top] = i;

}

for(int j = 1;j <= top; ++j) {

if(pri[j] \* i > N) break;

vis[pri[j] \* i] = true;

if(!(i % pri[j])) {

g[i \* pri[j]] = g[i] \* pri[j] % mod;

break;

}

else g[pri[j] \* i] = g[i] \* g[pri[j]] % mod;

}

}

for(int i = 1;i <= N; ++i) g[i] = (g[i] + g[i - 1]) % mod;

}

ll sum(ll x) {return (x \* (x + 1) >> 1) % mod;}

int main() {

get\_miu(); int T = read();

while(T--) {

n = read(), m = read();

if(n > m) swap(n, m);

ll ret = 0;

for(int i = 1, pos;i <= n; i = pos + 1) {

pos = min(n / (n / i), m / (m / i));

ret = (ret + (g[pos] - g[i - 1]) % mod \* sum(n / i) % mod \* sum(m / i) % mod) % mod;

}

printf("%lld\n", (ret + mod) % mod);

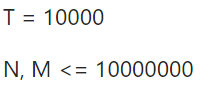
}

return 0;

}

**Problem 3**





//复杂度O(T\*√N)

#include <iostream>

#include <cstdio>

#include <cstring>

#include <algorithm>

#include <cmath>

using namespace std;

typedef long long ll;

const int N=1e7+5;

inline int read(){

char c=getchar();int x=0,f=1;

while(c<'0'||c>'9'){if(c=='-')f=-1; c=getchar();}

while(c>='0'&&c<='9'){x=x\*10+c-'0'; c=getchar();}

return x\*f;

}

int n,m;

bool notp[N];

int p[N],mu[N],g[N];

void sieve(){

mu[1]=1;

for(int i=2;i<N;i++){

if(!notp[i]) p[++p[0]]=i,mu[i]=-1,g[i]=1;

for(int j=1;j<=p[0]&&i\*p[j]<N;j++){

notp[i\*p[j]]=1;

if(i%p[j]==0){

mu[i\*p[j]]=0;

g[i\*p[j]]=mu[i];

break;

}

mu[i\*p[j]]=-mu[i];

g[i\*p[j]]=mu[i]-g[i];

}

}

for(int i=1;i<N;i++) g[i]+=g[i-1];

}

ll cal(int n,int m){

if(n>m) swap(n,m);

ll ans=0;int r;

for(int i=1;i<=n;i=r+1){

r=min(n/(n/i),m/(m/i));

ans+=(ll)(g[r]-g[i-1])\*(n/i)\*(m/i);

}

return ans;

}

int main(int argc, const char \* argv[]) {

sieve();

int T=read();

while(T--){

n=read();m=read();

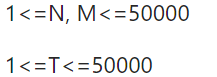
printf("%lld\n",cal(n,m));

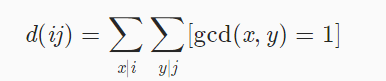
}

return 0;

}

**Problem 4**

// 

#include <cstdio>

#include <algorithm>

const int N=5e4+5;

int tot,mu[N],p[N];

long long s[N];

bool flg[N];

void init() {

mu[1]=1;

for(int i=2;i<=5e4;++i) {

if(!flg[i]) p[++tot]=i,mu[i]=-1;

for(int j=1;j<=tot&&i\*p[j]<=5e4;++j) {

flg[i\*p[j]]=1;

if(i%p[j]==0) {

mu[i\*p[j]]=0;

break;

} else {

mu[i\*p[j]]=-mu[i];

}

}

}

for(int i=1;i<=5e4;++i) mu[i]+=mu[i-1];

for(int x=1;x<=5e4;++x) {

long long res=0;

for(inti=1,j;i<=x;i=j+1) j=x/(x/i),res+=1LL\*(j-i+1)\*(x/i);

s[x]=res;

}

}

int main() {

init();

int T;

for(scanf("%d",&T);T--;) {

int n,m;

scanf("%d%d",&n,&m);

if(n>m) n^=m^=n^=m;

long long ans=0;

for(int i=1,j;i<=n;i=j+1) {

j=std::min(n/(n/i),m/(m/i));

ans+=1LL\*(mu[j]-mu[i-1])\*s[n/i]\*s[m/i];

}

printf("%lld\n",ans);

}

return 0;

}

#计算几何

### 基本操作（点、线、圆）

#include<bits/stdc++.h>

using namespace std;

const int maxn = 110;

const int INF = 0x7fffffff;

#define EPS (1e-10)

#define equals(a,b) (fabs((a) - (b)) < EPS)

// 点类

class Point {

public :

double x, y;

Point() {};

Point(double x, double y) :x(x), y(y) {}

Point operator + (Point p) { return Point(x + p.x, y + p.y); }

Point operator - (Point p) { return Point(x - p.x, y - p.y); }

Point operator \* (double a) { return Point(x \* a, y \* a); }

Point operator / (double a) { return Point(x / a, y / a); }

bool operator < (const Point &p) const {

return x != p.x ? x < p.x : y < p.y;

}

bool operator == (const Point &p) const {

return fabs(x - p.x) < EPS && fabs(y - p.y) < EPS;

}

};

// 线段类

class Segment {

public:

Point p1, p2;

Segment() {};

Segment(Point p1, Point p2) :p1(p1), p2(p2) {};

};

// 圆类

class Circle {

public:

Point c;

double r;

Circle() {};

Circle(Point c, double r) :c(c), r(r) {}

};

// 定义向量

typedef Point Vector;

// 定义直线

typedef Segment Line;

// 定义多边形

typedef vector<Point> Polygon;

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**点、向量**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

//求距离原点的距离

double norm(Point p) { return p.x \* p.x + p.y \* p.y; }

double abs(Point p) { return sqrt(norm(p)); }

// 向量的内积

double dot(Point a, Point b) {

return a.x \* b.x + a.y \* b.y;

}

// 向量的叉积

double cross(Point a, Point b) {

return a.x \* b.y - a.y \* b.x;

}

// 向量a，b是否正交 <==> 内积为0

bool isOrthogonal(Vector a, Vector b) {

return equals(dot(a, b), 0.0);

}

bool isOrthogonal(Point a1, Point a2, Point b1, Point b2) {

return equals(dot(a1 - a2, b1 - b2), 0.0);

}

// 向量a，b是否平行 <==> 外积为0

bool isParallel(Vector a, Vector b) {

return equals(cross(a, b), 0.0);

}

bool isParallel(Point a1, Point a2, Point b1, Point b2) {

return equals(cross(a1 - a2, b1 - b2), 0.0);

}

// 点p在线段s上的投影

Point project(Segment s, Point p) {

Vector base = s.p2 - s.p1;

double r = dot(p - s.p1, base) / norm(base);

return s.p1 + base \* r ;

}

//以线段s为对称轴与点p成线对称的点

Point reflect(Segment s, Point p) {

return p + (project(s, p) - p) \* 2.0;

}

// 点a到点b的距离

double getDistance(Point a, Point b) {

return abs(a - b);

}

// 线段l和点p的距离

double getDistanceLP(Line l, Point p) {

return abs( cross(l.p2 - l.p1, p - l.p1) / abs(l.p2 - l.p1) );

}

// 线段s与点p的距离

double getDistanceSP(Segment s, Point p) {

if (dot(s.p2 - s.p1, p - s.p1) < 0.0)

return abs(p - s.p1);

if (dot(s.p1 - s.p2, p - s.p2) < 0.0)

return abs(p - s.p2);

return getDistanceLP(s, p);

}

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**线段**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

// 线段s1，s2是否正交 <==> 内积为0

bool isOrthogonal(Segment s1, Segment s2) {

return equals(dot(s1.p2 - s1.p1, s2.p2 - s2.p1), 0.0);

}

// 线段s1，s2是否平行 <==> 外积为0

bool isParallel(Segment s1, Segment s2) {

return equals(cross(s1.p2 - s1.p1, s2.p2 - s2.p1), 0.0);

}

// 逆时针方向ccw（Counter-Clockwise）

static const int COUNTER\_CLOCKWISE = 1;

static const int CLOCKWISE = -1;

static const int ONLINE\_BACK = 2;

static const int ONLINE\_FRONT = -2;

static const int ON\_SEGMENT = 0;

int ccw(Point p0, Point p1, Point p2) {

Vector a = p1 - p0;

Vector b = p2 - p0;

if (cross(a, b) > EPS) return COUNTER\_CLOCKWISE;

if (cross(a, b) < -EPS) return CLOCKWISE;

if (dot(a, b) < -EPS) return ONLINE\_BACK;

if (norm(a) < norm(b)) return ONLINE\_FRONT;

return ON\_SEGMENT;

}

// 判断线段p1p2和线段p3p4是否相交

bool intersect(Point p1, Point p2, Point p3, Point p4) {

return (ccw(p1, p2, p3) \* ccw(p1, p2, p4) <= 0 &&

ccw(p3, p4, p1) \* ccw(p3, p4, p2) <= 0);

}

//判断线段s1和s2是否相交

bool intersect(Segment s1, Segment s2) {

return intersect(s1.p1, s1.p2, s2.p1, s2.p2);

}

// 线段s1和线段s2的距离

double getDistance(Segment s1, Segment s2) {

// 相交

if (intersect(s1, s2))

return 0.0;

return min(min(getDistanceSP(s1, s2.p1), getDistanceSP(s1, s2.p2)),

min(getDistanceSP(s2, s1.p1), getDistanceSP(s2, s1.p2)));

}

// 线段s1与线段s2的交点

Point getCrossPoint(Segment s1, Segment s2) {

Vector base = s2.p2 - s2.p1;

double d1 = abs(cross(base, s1.p1 - s2.p1));

double d2 = abs(cross(base, s1.p2 - s2.p1));

double t = d1 / (d1 + d2);

return s1.p1 + (s1.p2 - s1.p1) \* t;

}

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**圆**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

// 圆c和直线l的交点

pair<Point, Point> getCrossPoints(Circle c, Line l) {

Vector pr = project(l, c.c);

Vector e = (l.p2 - l.p1) / abs(l.p2 - l.p1);

double base = sqrt(c.r \* c.r - norm(pr - c.c));

return make\_pair(pr + e \* base, pr - e \* base);

}

// 圆c1和圆c2的交点

double arg(Vector p) { return atan2(p.y, p.x); }

Vector polar(double a, double r) { return Point(cos(r) \* a, sin(r) \* a); }

pair<Point, Point> getCrossPoints(Circle c1, Circle c2) {

double d = abs(c1.c - c2.c);

double a = acos((c1.r \* c1.r + d \* d - c2.r \* c2.r) / (2 \* c1.r \* d));

double t = arg(c2.c - c1.c);

return make\_pair(c1.c + polar(c1.r, t + a), c1.c + polar(c1.r, t - a));

}

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**多边形**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

// 点的内包

/\*

IN 2

ON 1

OUT 0

\*/

int contains(Polygon g, Point p) {

int n = g.size();

bool x = false;

for (int i = 0; i < n; i++) {

Point a = g[i] - p, b = g[(i + 1) % n] - p;

if (abs(cross(a, b)) < EPS && dot(a, b) < EPS) return 1;

if (a.y > b.y) swap(a, b);

if (a.y < EPS && EPS < b.y && cross(a, b) > EPS)

x = !x;

}

return (x ? 2 : 0);

}

int cmp(Point A, Point B) //竖直排序

{

return (A.y<B.y || (A.y == B.y&&A.x<B.x));

}

// 凸包

Polygon andrewScan(Polygon s) {

Polygon u, l;

int len = s.size();

if (len < 3) return s;

// 以x，y为基准升序排序

sort(s.begin(), s.end());

// 将x值最小的两个点添加到u

u.push\_back(s[0]);

u.push\_back(s[1]);

// 将x值最大的两个点添加到l

l.push\_back(s[len - 1]);

l.push\_back(s[len - 2]);

// 构建凸包上部

for (int i = 2; i < len; i++) {

for (int j = u.size(); j >= 2 && ccw(u[j - 2], u[j - 1], s[i]) >= 0; j--) {

u.pop\_back();

}

u.push\_back(s[i]);

}

// 构建凸包下部

for (int i = len - 3; i >= 0; i--) {

for (int j = l.size(); j >= 2 && ccw(l[j - 2], l[j - 1], s[i]) >= 0; j--) {

l.pop\_back();

}

l.push\_back(s[i]);

}

reverse(l.begin(), l.end());

for (int i = u.size() - 2; i >= 1; i--)

l.push\_back(u[i]);

return l;

}

### 极角排序

//利用atan2函数

//atan2(y,x)，表示(x,y)这个点与原点连线，这条线与x轴正半轴的夹角，这里的这个极角的范围是[-π,π]的，

//一二象限为正，三四象限为负。所以我们从小到大排完序后，实际上是 第三象限<第四象限<第一象限<第二象限

struct point{

int x,y;

double angle;

inline bool operator < (const point &t)const{

return angle<t.angle;

}

}a[N];

for(int i=1;i<=n;i++)

{

read(a[i].x);read(a[i].y);//输入

a[i].angle=atan2(a[i].y,a[i].x);

}

sort(a+1,a+n+1,cmp);

//利用叉积

//利用叉积的正负来判断两个向量的相对方向

//注意：如果取的点不是边角的点，那么需要先按照象限排序，

//再按照极角排序，否则排序会出错，因为叉积大小正负是相对的

struct point{

int x,y;

double angle;

inline int operator \* (const node &t)const{//叉积

return a.x\*b.y-a.y\*b.x;

}

};

inline bool cmp(point a,point b)

{

return a\*b>0;

}

for(int i=1;i<=n;i++)

{

read(a[i].x);read(a[i].y);//输入

}

sort(a+1,a+n+1,cmp);

//先按象限从小到大排 再按极角从小到大排

int Quadrant(point a)

{

if(a.x>0&&a.y>=0)

return 1;

if(a.x<=0&&a.y>0)

return 2;

if(a.x<0&&a.y<=0)

return 3;

if(a.x>=0&&a.y<0)

return 4;

}

bool cmp(point a,point b)

{

if(Quadrant(a)==Quadrant(b))

return cmp(a,b);

else

return Quadrant(a)<Quadrant(b);

}

**例**：**（极角排序）**

***Space Ant***

struct point{

int x,y,id;

}P[N];

int cross(point a,point b,point c)//ab×ac

{

return(b.x-a.x)\*(c.y-a.y)-(b.y-a.y)\*(c.x-a.x);

}

double dist(point a,point b)

{

return sqrt(((b.x-a.x)\*(b.x-a.x)+(b.y-a.y)\*(b.y-a.y)\*1.0));

}

Int pos;

bool cmp(point a,point b)//极角排序

{

int tmp=cross(P[pos],a,b);

if(tmp==0)

return dist(P[pos],a)<dist(P[pos],b);

else if(tmp>0)

return 1;

else

return 0;

}

int main()

{

scanf("%d",&n);

for(int i=1;i<=n;i++)

{

scanf("%d%d%d",&P[i].id,&P[i].x,&P[i].y);

if(P[i].y<P[1].y||(P[i].y==P[1].y&&P[i].x<P[1].x))

swap(P[1],P[i]);//寻找左下角的点作为起点

}

pos=1;

for(int i=2;i<=n;i++)

{

sort(P+i,P+n+1,cmp);//不断进行极角排序

pos++;

}

printf("%d",n);

for(int i=1;i<=n;i++)

printf(" %d",P[i].id);//

printf("\n");

}

### 二维凸包

const int N = 30;

struct point{

double x,y;

}P[N];

int cross(point a,point b,point c)//ab×ac

{

return(b.x-a.x)\*(c.y-a.y)-(b.y-a.y)\*(c.x-a.x);

}

double dist(point a,point b)

{

return sqrt(((b.x-a.x)\*(b.x-a.x)+(b.y-a.y)\*(b.y-a.y))\*1.0);

}

bool cmp(point a,point b)//极角排序

{

int tmp=cross(P[0],a,b);

if(tmp==0)

return dist(P[0],a)<dist(P[0],b);

else if(tmp>0)

return 1;

else

return 0;

}

int stack[N],top,n,cnt;//stack存凸包的点

double totdis;

double totcost,totlen;

void Graham()

{

totdis=0;

point p0=P[0];//从0开始输入

int k=0;

for(int i=1;i<cnt;i++)

{

if(p0.y>P[i].y||(p0.y==P[i].y&&p0.x>P[i].x))

{

p0=P[i];

k=i;

}

}

swap(P[0],P[k]);

sort(P+1,P+cnt,cmp);//从下一个开始排序

if(cnt==1)

{

top=1;

stack[0]=0;

return;

}

if(cnt==2)

{

top=2;

stack[0]=0;

stack[1]=1;

return;

}

stack[0]=0;

stack[1]=1;

top=2;

for(int i=2;i<cnt;i++)

{

while(top>1&&cross(P[stack[top-2]],P[stack[top-1]],P[i])<=0)

top--;

stack[top++]=i;

}

}

### 最大空凸包

//计算最大空凸包面积

#include<bits/stdc++.h>

using namespace std;

#define LL long long

#define mod 1000000007

typedef struct Point

{

int x, y;

Point() {}

Point(int x, int y): x(x),y(y) {}

Point operator + (const Point &b) const { return Point(x+b.x,y+b.y); }

Point operator - (const Point &b) const { return Point(x-b.x,y-b.y); }

int operator \* (const Point &b) const { return x\*b.y-y\*b.x; }

int len() const { return x\*x+y\*y; }

int operator < (const Point &a) const

{

if((\*this)\*a>0 || (\*this)\*a==0 && len()<a.len())

return 1;

return 0;

}

}Point;

int n;

Point s[122], p[122];

int dp[122][122];

int Jud(int m)

{

int ans, i, j, now, k, flag, S;

memset(dp, 0, sizeof(dp));

ans = 0;

for(i=2;i<=m;i++)

{

now = i-1;

while(now>=1 && p[i]\*p[now]==0)

now--;

flag = 0;

if(now==i-1)

flag = 1;

while(now>=1)

{

S = p[now]\*p[i];

k = now-1;

while(k>=1 && (p[now]-p[i])\*(p[k]-p[now])>0)

k--;

if(k>=1)

S += dp[now][k];

if(flag)

dp[i][now] = S;

ans = max(ans, S);

now = k;

}

if(flag==0)

continue;

for(j=1;j<=i-1;j++)

dp[i][j] = max(dp[i][j],dp[i][j-1]);

}

return ans;

}

int main()

{

int T, i, j, m, ans;

scanf("%d", &T);

while(T--)

{

scanf("%d", &n);

for(i=1;i<=n;i++)

scanf("%d%d", &s[i].x, &s[i].y);

ans = 0;

for(i=1;i<=n;i++)

{

m = 0;

for(j=1;j<=n;j++)

{

if(s[j].y>s[i].y || s[j].y==s[i].y && s[j].x>=s[i].x)

p[++m] = s[j]-s[i];

}

sort(p+1, p+m+1);

ans = max(ans, Jud(m));

}

printf("%.1f\n", ans/2.0);

}

return 0;

}

### 三维凸包

#include<iostream>

#include<cmath>

#include<cstring>

#include<cstdlib>

#include<algorithm>

#include <stdio.h>

using namespace std;

const int MAXN=1001;

const int N = 500;

const double EPS=1e-8;

int g[MAXN][MAXN];

struct Point{

double x,y,z;

Point(){}

Point(double xx,double yy,double zz)

:x(xx),y(yy),z(zz){}

Point operator -(const Point p1){//两向量之差

return Point(x-p1.x,y-p1.y,z-p1.z);

}

Point operator \*(Point p){//叉乘

return Point(y\*p.z-z\*p.y,z\*p.x-x\*p.z,x\*p.y-y\*p.x);

}

double operator ^(Point p){//点乘

return (x\*p.x+y\*p.y+z\*p.z);

}

};

struct CH3D{

struct face{

int a,b,c;//表示凸包一个面上三个点的编号

bool ok;//表示该面是否属于最终凸包中的面

};

int n;//初始顶点数

Point P[MAXN];//初始顶点

int num; //凸包表面的三角形数

face F[8\*MAXN];

//int g[MAXN][MAXN];//凸包表面的三角形

double vlen(Point a){//向量长度

return sqrt(a.x\*a.x+a.y\*a.y+a.z\*a.z);

}

Point cross(const Point &a, const Point &b, const Point &c){//叉乘

return Point((b.y-a.y)\*(c.z-a.z)-(b.z-a.z)\*(c.y-a.y),-((b.x-a.x)\*(c.z-a.z)-(b.z-a.z)\*(c.x-a.x)),(b.x-a.x)\*(c.y-a.y)-(b.y-a.y)\*(c.x-a.x));

}

double area(Point a,Point b,Point c){//三角形面积\*2

return vlen((b-a)\*(c-a));

}

double volume(Point a,Point b,Point c,Point d){//四面体有向体积\*6

return (b-a)\*(c-a)^(d-a);

}

double dblcmp(Point &p,face &f){//正:点在面同向

Point m=P[f.b]-P[f.a];

Point n=P[f.c]-P[f.a];

Point t=p-P[f.a];

return (m\*n)^t;

}

void deal(int p,int a,int b){

int f=g[a][b];

face add;

if(F[f].ok){

if(dblcmp(P[p],F[f])>EPS){

dfs(p,f);

}

else {

add.a=b;

add.b=a;

add.c=p;

add.ok=1;

g[p][b]=g[a][p]=g[b][a]=num;

F[num++]=add;

}

}

}

void dfs(int p,int now){

F[now].ok=0;

deal(p,F[now].b,F[now].a);

deal(p,F[now].c,F[now].b);

deal(p,F[now].a,F[now].c);

}

bool same(int s,int t){

Point &a=P[F[s].a];

Point &b=P[F[s].b];

Point &c=P[F[s].c];

return fabs(volume(a,b,c,P[F[t].a]))<EPS && fabs(volume(a,b,c,P[F[t].b]))<EPS

&& fabs(volume(a,b,c,P[F[t].c]))<EPS;

}

void pretreat(){//构建三维凸包

int i,j,tmp;

face add;

bool flag;

num=0;

if(n<4) return;

flag=true;

for(i=1;i<n;i++){//此段是为了保证前四个点不共面,若以保证,则可去掉

if(vlen(P[0]-P[i])>EPS){

swap(P[1],P[i]);

flag=false; break;

}

}

if(flag) return;

flag=true;

for(i=2;i<n;i++){//使前三点不共线

if(vlen((P[0]-P[1])\*(P[1]-P[i]))>EPS){

swap(P[2],P[i]);

flag=false; break;

}

}

if(flag) return;

flag=true;

for(i=3;i<n;i++){//使前四点不共面

if(fabs((P[0]-P[1])\*(P[1]-P[2])^(P[0]-P[i]))>EPS){

swap(P[3],P[i]);

flag=false;

break;

}

}

if(flag) return;

for(i=0;i<4;i++){

add.a=(i+1)%4;

add.b=(i+2)%4;

add.c=(i+3)%4;

add.ok=true;

if(dblcmp(P[i],add)>0)

swap(add.b,add.c);

g[add.a][add.b]=g[add.b][add.c]=g[add.c][add.a]=num;

F[num++]=add;

}

for(i=4;i<n;i++){

for(j=0;j<num;j++){

if(F[j].ok && dblcmp(P[i],F[j])>EPS){

dfs(i,j);

break;

}

}

}

tmp=num;

for(i=num=0;i<tmp;i++){

if(F[i].ok) F[num++]=F[i];

}

}

double area(){//表面积

double res=0.0;

if(n==3){

Point p=cross(P[0],P[1],P[2]);

res=vlen(p)/2.0;

return res;

}

for(int i=0;i<num;i++)

res+=area(P[F[i].a],P[F[i].b],P[F[i].c]);

return res/2.0;

}

double volume(){//体积

double res=0.0;

Point tmp(0,0,0);

for(int i=0;i<num;i++)

res+=volume(tmp,P[F[i].a],P[F[i].b],P[F[i].c]);

return fabs(res/6.0);

}

int triangle(){//表面三角形个数

return num;

}

int polygon\_bug(){//表面多边形个数

int i,j,res,flag;

for(i=res=0;i<num;i++){

flag=1;

for(j=0;j<i;j++){

if(same(i,j)){

flag=0; break;

}

res+=flag;

}

}

return res;

}

int polygon(){//表面多边形个数

int i,j,res,flag;

for(i=res=0;i<num;i++){

flag=1;

for(j=0;j<i;j++)

if(same(i,j)){

flag=0; break;

}

res+=flag;

}

return res;

}

Point getcent(){//求凸包质心

Point ans(0,0,0),temp=P[F[0].a];

double v = 0.0,t2;

for(int i=0;i<num;i++){

if(F[i].ok == true){

Point p1=P[F[i].a],p2=P[F[i].b],p3=P[F[i].c];

t2 = volume(temp,p1,p2,p3)/6.0;//体积大于0，也就是说，点 temp 不在这个面上

if(t2>0){

ans.x += (p1.x+p2.x+p3.x+temp.x)\*t2;

ans.y += (p1.y+p2.y+p3.y+temp.y)\*t2;

ans.z += (p1.z+p2.z+p3.z+temp.z)\*t2;

v += t2;

}

}

}

ans.x /= (4\*v); ans.y /= (4\*v); ans.z /= (4\*v);

return ans;

}

double function(Point fuck){//点到凸包(平面)上的最近距离（枚举每个面到这个点的距离）

double min=99999999;

for(int i=0;i<num;i++){

if(F[i].ok==true){

Point p1=P[F[i].a] , p2=P[F[i].b] , p3=P[F[i].c];

double a = ( (p2.y-p1.y)\*(p3.z-p1.z)-(p2.z-p1.z)\*(p3.y-p1.y) );

double b = ( (p2.z-p1.z)\*(p3.x-p1.x)-(p2.x-p1.x)\*(p3.z-p1.z) );

double c = ( (p2.x-p1.x)\*(p3.y-p1.y)-(p2.y-p1.y)\*(p3.x-p1.x) );

double d = ( 0-(a\*p1.x+b\*p1.y+c\*p1.z) );

double temp = fabs(a\*fuck.x+b\*fuck.y+c\*fuck.z+d)/sqrt(a\*a+b\*b+c\*c);

if(temp<min)min = temp;

}

}

return min;

}

double ptoface(Point p,int i)//求点(确定)某个平面(确定)的距离

{ return fabs(volume(P[F[i].a],P[F[i].b],P[F[i].c],p)/vlen((P[F[i].b]-P[F[i].a])\*(P[F[i].c]-P[F[i].a])));

}

};

int main()

{

int i,n,m;

while(scanf("%d",&n),n){

CH3D hull;

memset(g,0,sizeof(g));

memset(hull.P,0,sizeof(hull.P));

hull.n = n;

for(i=0;i<hull.n;i++){

scanf("%lf%lf%lf",&hull.P[i].x,&hull.P[i].y,&hull.P[i].z);

}

hull.pretreat();

scanf("%d",&m);//询问m个点

Point temp;

for(i=0;i<m;i++){

scanf("%lf%lf%lf",&temp.x,&temp.y,&temp.z);

printf("%.4lf\n",hull.function(temp));

//**点到面的最短距离**

}

}

return 0;

}

// hull.polygon();**输出凸包表面多边形的个数**

/\***求重心(质心)到每个面的最小距离(枚举)**

Point res=hull.getcent();

double minn=1e20;

for(int i=0;i<hull.num;i++)

{

minn=min(minn,hull.ptoface(res,i));

}

printf("%.3f\n",minn);

\*/

//hull.area()**输出凸包表面积**

### 半平面交

const int N = 1e3;

const double EPS = 1e-5;

int T,n;

typedef struct Grid{

double x,y;

Grid(double a=0,double b=0){x=a;y=b;}

}Point,Vector;

Vector operator - (Point a,Point b)//ab向量

{

return Vector(b.x-a.x,b.y-a.y);

}

double operator ^ (Vector a,Vector b)//a×b

{

return a.x\*b.y-a.y\*b.x;

}

struct Line{

Point s,e;

Line(){}

Line(Point a,Point b){s=a;e=b;}

};

Point p[N];

Line line[N],que[N];

//得到极角角度

double getangle(Vector a)

{

return atan2(a.y,a.x);

}

//得到极角角度

double getangle(Line a)

{

return atan2(a.e.y-a.s.y,a.e.x-a.s.x);

}

//排序：极角小的排前面,最左边的排在最后面

bool cmp(Line a,Line b)

{

Vector va=a.e-a.s,vb=b.e-b.s;

double A=getangle(va),B=getangle(vb);

if(fabs(A-B)<EPS)

return ((va)^(b.e-a.s))>=0;//等号可不加

return A<B;

}

**//得到两直线相交的交点**

**Point getintersectpoint(Line a,Line b)**

**{**

**double a1=a.s.y-a.e.y,b1=a.e.x-a.s.x,c1=a.s.x\*a.e.y-a.e.x\*a.s.y;**

**double a2=b.s.y-b.e.y,b2=b.e.x-b.s.x,c2=b.s.x\*b.e.y-b.e.x\*b.s.y;**

**return Point((c1\*b2-c2\*b1)/(a2\*b1-a1\*b2),(a2\*c1-a1\*c2)/(a1\*b2-a2\*b1));**

**}**

//判断b,c的交点是否在a的右边

bool onright(Line a,Line b,Line c)

{

Point o=getintersectpoint(b,c);

if(((a.e-a.s)^(o-a.s))<0)return 1;

return 0;

}

bool halfplaneintersection()

{

sort(line,line+n,cmp);

int head=0,tail=0,cnt=0;//模拟双端队列

//去重，极角相同时取最后一个

for(int i=0;i<n-1;i++)

{

if(fabs(getangle(line[i])-getangle(line[i+1]))<EPS)

continue;

line[cnt++]=line[i];

}

line[cnt++]=line[n-1];

for(int i=0;i<cnt;i++)

{//判断新加入的直线产生的影响 while(tail-head>1&&onright(line[i],que[tail-1],que[tail-2]))

tail--; while(tail-head>1&&onright(line[i],que[head],que[head+1]))

head++;

que[tail++]=line[i];

}

//最后判断最先加入的直线和最后的直线的影响

while(tail-head>1&&onright(que[head],que[tail-1],que[tail-2]))

tail--;

while(tail-head>1&&onright(que[tail-1],que[head],que[head+1]))

head++;

if(tail-head<3)

return 0;

return 1;

}

//判断输入点的顺序，如果面积小于0，说明输入的点为逆时针，否则为顺时针

bool judge()

{

double ans=0;

for(int i=1;i<n-1;i++)

ans+=(p[i]-p[0])^(p[i+1]-p[0]);

return ans<0;

}

int main()

{

scanf("%d",&T);

while(T--)

{

scanf("%d",&n);

for(int i=n-1;i>=0;i--)

scanf("%lf%lf",&p[i].x,&p[i].y);

if(judge())//判断输入顺序，保证逆时针连边

{

for(int i=0;i<n;i++)

line[i]=Line(p[(i+1)%n],p[i]);

}

else

{

for(int i=0;i<n;i++)

line[i]=Line(p[i],p[(i+1)%n]);

}

if(halfplaneintersection())

printf("YES\n");

else

printf("NO\n");

}

}

### 凸多边形最大内切圆

//输入边

const int N = 1e3;

const double EPS = 1e-6;

int T,n;

typedef struct Grid{

double x,y;

Grid(double a=0,double b=0){x=a;y=b;}

}Point,Vector;

Vector operator - (Point a,Point b)//ab向量

{

return Vector(b.x-a.x,b.y-a.y);

}

double operator ^ (Vector a,Vector b)//a×b

{

return a.x\*b.y-a.y\*b.x;

}

struct Line{

Point s,e;

Line(){}

Line(Point a,Point b){s=a;e=b;}

};

Point p[N];

Line line[N],que[N];

//得到极角角度

double getangle(Vector a)

{

return atan2(a.y,a.x);

}

//得到极角角度

double getangle(Line a)

{

return atan2(a.e.y-a.s.y,a.e.x-a.s.x);

}

//排序：极角小的排前面,最左边的排在最后面

bool cmp(Line a,Line b)

{

Vector va=a.e-a.s,vb=b.e-b.s;

double A=getangle(va),B=getangle(vb);

if(fabs(A-B)<EPS)

return ((va)^(b.e-a.s))>0;

return A<B;

}

//得到两直线相交的交点

Point getintersectpoint(Line a,Line b)

{

double a1=a.s.y-a.e.y,b1=a.e.x-a.s.x,c1=a.s.x\*a.e.y-a.e.x\*a.s.y;

double a2=b.s.y-b.e.y,b2=b.e.x-b.s.x,c2=b.s.x\*b.e.y-b.e.x\*b.s.y;

return Point((c1\*b2-c2\*b1)/(a2\*b1-a1\*b2),(a2\*c1-a1\*c2)/(a1\*b2-a2\*b1));

}

//判断b,c的交点是否在a的右边

bool onright(Line a,Line b,Line c)

{

Point o=getintersectpoint(b,c);

if(((a.e-a.s)^(o-a.s))<0)return 1;

return 0;

}

bool halfplaneintersection()

{

sort(line,line+n,cmp);

int head=0,tail=0,cnt=0;//模拟双端队列

//去重，极角相同时取最后一个

for(int i=0;i<n-1;i++)

{

if(fabs(getangle(line[i])-getangle(line[i+1]))<EPS)

continue;

line[cnt++]=line[i];

}

line[cnt++]=line[n-1];

for(int i=0;i<cnt;i++)

{

//判断新加入的直线产生的影响

while(tail-head>1&&onright(line[i],que[tail-1],que[tail-2]))

tail--;

while(tail-head>1&&onright(line[i],que[head],que[head+1]))

head++;

que[tail++]=line[i];

}

//最后判断最先加入的直线和最后的直线的影响

while(tail-head>1&&onright(que[head],que[tail-1],que[tail-2]))

tail--;

while(tail-head>1&&onright(que[tail-1],que[head],que[head+1]))

head++;

if(tail-head<3)

return 0;

return 1;

}

//判断输入点的顺序，如果面积小于0，说明输入的点为逆时针，否则为顺时针

bool judge()

{

double ans=0;

for(int i=1;i<n-1;i++)

ans+=(p[i]-p[0])^(p[i+1]-p[0]);

return ans<0;

}

double dis(Point a,Point b)

{

return sqrt((a.x-b.x)\*(a.x-b.x)+(a.y-b.y)\*(a.y-b.y));

}

Line pl[N];

void change(int n,double h)//将所有边往里移动h

{

for(int i=0;i<n;i++)

{

double rat=h/dis(pl[i].s,pl[i].e);

line[i].s.x=pl[i].s.x-(pl[i].e.y-pl[i].s.y)\*rat;

line[i].s.y=pl[i].s.y+(pl[i].e.x-pl[i].s.x)\*rat;

line[i].e.x=pl[i].e.x-(pl[i].e.y-pl[i].s.y)\*rat;

line[i].e.y=pl[i].e.y+(pl[i].e.x-pl[i].s.x)\*rat;

}

}

int main()

{

while(~scanf("%d", &n)&&n)

{

for (int i = 0; i <n; i++)

scanf("%lf%lf", &p[i].x, &p[i].y);

for (int i = 0; i < n; i++)

pl[i] = Line(p[i], p[(i + 1)%n]);

double l=0,r=2e4,mid,ans;

while(r-l>EPS)

{

mid=(l+r)/2;

change(n,mid);

if (halfplaneintersection())

l=mid;

else

r=mid;

}

printf("%.6f\n",l);

}

}

###最小覆盖球

1. 模拟淬火

#include<bits/stdc++.h>

const int MAXN = 1000;

const double eps = 1e-8;

int n;

struct Point {

double x, y, z;

} myPoint[MAXN], op;

double GetDis(const Point &a, const Point &b) {

return sqrt((a.x - b.x) \* (a.x - b.x) + (a.y - b.y) \* (a.y - b.y) +

(a.z - b.z) \* (a.z - b.z));

}

double Solve() {

double ret = 1e30, delta = 100000.0;

double maxDis, tempDis;

int i, id;

while (delta > eps) {

id = 0;

maxDis = GetDis(op, myPoint[id]);

for (i = 1; i < n; i++) {

tempDis = GetDis(op, myPoint[i]);

if (tempDis > maxDis) {

maxDis = tempDis;

id = i;

}

}

ret = min(ret, maxDis);

op.x += (myPoint[id].x - op.x) / maxDis \* delta;

op.y += (myPoint[id].y - op.y) / maxDis \* delta;

op.z += (myPoint[id].z - op.z) / maxDis \* delta;

delta \*= 0.98;

}

return ret;

}

int main() {

int i;

scanf("%d", &n);

for (i = 0; i < n; i++) {

int x, y, z;

scanf("%d%d%d", &x, &y, &z);

myPoint[i].x = x, myPoint[i].y = y, myPoint[i].z = z;

}

op.x = op.y = op.z = -100000.0;

printf("%.15f\n", Solve());

return 0;

}

1. 三分

#include<bits/stdc++.h>

using namespace std;

typedef long long ll;

const int maxn = 1e5 + 10;

struct node

{

double x, y, z;

}a[maxn];

int n;

double dis(double x, double y, double z)

{

double ret = 0;

for(int i = 0; i < n; ++i)

{

ret = max(ret, sqrt((a[i].x - x) \* (a[i].x - x) + (a[i].y - y) \* (a[i].y - y) + (a[i].z - z) \* (a[i].z - z)));

}

return ret;

}

double slove(double c[], int num)

{

if(num >= 3)

return dis(c[0], c[1], c[2]);

double l = -1e5, r = 1e5;

double midl, midr, ans;

while(r - l > 1e-6)

{

midl = l + (r - l) / 3;

midr = r - (r - l) / 3;

c[num] = midl;

double ans1 = slove(c, num + 1);

c[num] = midr;

double ans2 = slove(c, num + 1);

if(ans1 > ans2)

{

ans = ans2;

l = midl;

}

else

{

ans = ans1;

r = midr;

}

}

return ans;

}

int main()

{

scanf("%d", &n);

for(int i = 0; i < n; ++i)

scanf("%lf%lf%lf", &a[i].x, &a[i].y, &a[i].z);

double c[4];

double ans = slove(c, 0);

printf("%.10f\n", ans);

return 0;

}

1. 其它

#include<bits/stdc++.h>

using namespace std;

typedef long long ll;

const double eps = 1e-6;

struct Tpoint

{

double x,y,z;

};

int npoint,nouter;

Tpoint pt[200000],outer[4],res;

double radius,tmp;

inline double dist(Tpoint p1,Tpoint p2)

{

double dx=p1.x-p2.x,dy=p1.y-p2.y,dz=p1.z-p2.z;

return (dx\*dx+dy\*dy+dz\*dz);

}

inline double dot(Tpoint p1,Tpoint p2)

{

return p1.x\*p2.x+p1.y\*p2.y+p1.z\*p2.z;

}

void ball()

{

Tpoint q[3];

double m[3][3],sol[3],L[3],det;

int i,j;

res.x=res.y=res.z=radius=0;

switch(nouter)

{

case 1:

res=outer[0];

break;

case 2:

res.x=(outer[0].x+outer[1].x)/2;

res.y=(outer[0].y+outer[1].y)/2;

res.z=(outer[0].z+outer[1].z)/2;

radius=dist(res,outer[0]);

break;

case 3:

for(int i=0; i<2; ++i)

{

q[i].x=outer[i+1].x-outer[0].x;

q[i].y=outer[i+1].y-outer[0].y;

q[i].z=outer[i+1].z-outer[0].z;

}

for(int i=0; i<2; ++i)

for(int j=0; j<2; ++j)

m[i][j]=dot(q[i],q[j])\*2;

for(int i=0; i<2; ++i)

sol[i]=dot(q[i],q[i]);

if(fabs(det=m[0][0]\*m[1][1]-m[0][1]\*m[1][0])<eps)

return;

L[0]=(sol[0]\*m[1][1]-sol[1]\*m[0][1])/det;

L[1]=(sol[1]\*m[0][0]-sol[0]\*m[1][0])/det;

res.x=outer[0].x+q[0].x\*L[0]+q[1].x\*L[1];

res.y=outer[0].y+q[0].y\*L[0]+q[1].y\*L[1];

res.z=outer[0].z+q[0].z\*L[0]+q[1].z\*L[1];

radius=dist(res,outer[0]);

break;

case 4:

for(int i=0; i<3; ++i)

{

q[i].x=outer[i+1].x-outer[0].x;

q[i].y=outer[i+1].y-outer[0].y;

q[i].z=outer[i+1].z-outer[0].z;

sol[i]=dot(q[i],q[i]);

}

for(int i=0; i<3; ++i)

for(int j=0; j<3; ++j)

m[i][j]=dot(q[i],q[j])\*2;

det=m[0][0]\*m[1][1]\*m[2][2]

+m[0][1]\*m[1][2]\*m[2][0]

+m[0][2]\*m[2][1]\*m[1][0]

-m[0][2]\*m[1][1]\*m[2][0]

-m[0][1]\*m[1][0]\*m[2][2]

-m[0][0]\*m[1][2]\*m[2][1];

if(fabs(det)<eps)return ;

for(int j=0; j<3; ++j)

{

for(int i=0; i<3; ++i)

m[i][j]=sol[i];

L[j]=(m[0][0]\*m[1][1]\*m[2][2]

+m[0][1]\*m[1][2]\*m[2][0]

+m[0][2]\*m[2][1]\*m[1][0]

-m[0][2]\*m[1][1]\*m[2][0]

-m[0][1]\*m[1][0]\*m[2][2]

-m[0][0]\*m[1][2]\*m[2][1]

)/det;

for(int i=0; i<3; ++i)

m[i][j]=dot(q[i],q[j])\*2;

}

res=outer[0];

for(int i=0; i<3; ++i)

{

res.x+=q[i].x\*L[i];

res.y+=q[i].y\*L[i];

res.z+=q[i].z\*L[i];

}

radius=dist(res,outer[0]);

}

}

void minball(int n)

{

ball();

if(nouter<4)

{

for(int i=0; i<n; ++i)

if(dist(res,pt[i])-radius>eps)

{

outer[nouter]=pt[i];

++nouter;

minball(i);

--nouter;

if(i>0)

{

Tpoint Tt=pt[i];

memmove(&pt[1],&pt[0],sizeof(Tpoint)\*i);

pt[0]=Tt;

}

}

}

}

double smallest\_ball()//求最小覆盖球的半径

{

radius=-1;

for(int i=0; i<npoint; ++i)

{

if(dist(res,pt[i])-radius>eps)

{

nouter=1;

outer[0]=pt[i];

minball(i);

}

}

return sqrt(radius);

}

int main()

{

scanf("%d",&npoint);//点个数

for(int i=0; i<npoint; ++i)

scanf("%lf%lf%lf",&pt[i].x,&pt[i].y,&pt[i].z);//点坐标

cout<<fixed<<setprecision(15)<<smallest\_ball()<<endl;//输出最小覆盖球半径

}

###最小圆覆盖

#include<bits/stdc++.h>

using namespace std;

const int MAXN=500+10;

const double eps=1e-8;

int n;

struct Point

{

double x,y;

}myPoint[MAXN],op;

double GetDis(const Point &a,const Point &b)

{

return sqrt((a.x-b.x)\*(a.x-b.x)+(a.y-b.y)\*(a.y-b.y));

}

void Solve()

{

double ret,delta=100.0;

double maxDis,tempDis;

int i,id;

while(delta>eps)

{

id=0;

maxDis=GetDis(op,myPoint[id]);

for(i=1;i<n;i++)

{

tempDis=GetDis(op,myPoint[i]);

if(tempDis>maxDis)

{

maxDis=tempDis;

id=i;

}

}

ret=maxDis;

op.x+=(myPoint[id].x-op.x)/maxDis\*delta;

op.y+=(myPoint[id].y-op.y)/maxDis\*delta;

delta\*=0.98;

}

printf("%.2lf %.2lf %.2lf\n",op.x,op.y,ret);

}

int main()

{

int i;

while(scanf("%d",&n)!=EOF)

{

if(0==n)break;

op.x=op.y=0;

for(i=0;i<n;i++)

{

scanf("%lf%lf",&myPoint[i].x,&myPoint[i].y);

op.x+=myPoint[i].x;

op.y+=myPoint[i].y;

}

op.x/=n;

op.x/=n;

Solve();

}

return 0;

}

### 最近平面点对

//复杂度 O(nlogn)

#include<bits/stdc++.h>

using namespace std;

const int N=1e6+10;

const int INF=2<<20;

int n,temp[N];

struct Point

{

double x,y;

}S[N];

bool cmp(const Point &a,const Point&b)

{

if(a.x==b.x)

return a.y<b.y;

else

return a.x<b.x;

}

bool cmps(const int &a,const int &b)

{

return S[a].y<S[b].y ;

}

double min(double a,double b)

{

return a<b?a:b;

}

double dist(int i,int j)

{

double x=(S[i].x-S[j].x)\*(S[i].x-S[j].x);

double y=(S[i].y-S[j].y)\*(S[i].y-S[j].y);

return sqrt(x+y);

}

double merge(int left,int right)

{

double d=INF;

if(left==right)

return d ;

if(left+1==right)

return dist(left,right);

int mid=left+right>>1;

double d1=merge(left,mid) ;

double d2=merge(mid+1,right) ;

d=min(d1,d2);

int i,j,k=0;

for(i=left;i<=right;i++)

if(fabs(S[mid].x-S[i].x)<=d)

temp[k++]=i;

sort(temp,temp+k,cmps);

for(i=0;i<k;i++)

for(j=i+1;j<k&&S[temp[j]].y-S[temp[i]].y<d;j++)

{

double d3=dist(temp[i],temp[j]);

if(d>d3)

d=d3;

}

return d;

}

int main()

{

scanf("%d",&n);

for(int i=0;i<n;i++)

scanf("%lf%lf",&S[i].x,&S[i].y);

sort(S,S+n,cmp);

printf("%.4lf\n",merge(0,n-1));

}

#常用操作

### 二维差分

void add(int x1,int y1,int x2,int y2)//差分

{

sum[x1][y1]++;

sum[x1][y2+1]--;

sum[x2+1][y1]--;

sum[x2+1][y2+1]++;

}

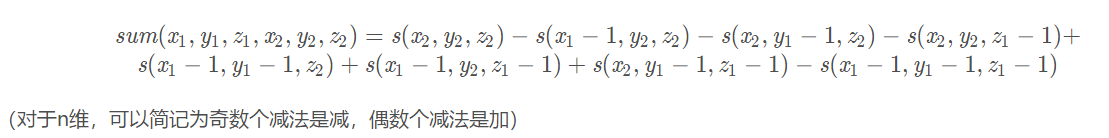
### 二维前缀和

for(int i=1;i<=n;i++)

for(int j=1;j<=n;j++)

sum[i][j]+=sum[i-1][j]+sum[i][j-1]-sum[i-1][j-1];

### 多维前缀和

****

### 离散化

stable\_sort(v.begin(),v.end());

v.erase(unique(v.begin(),v.end()),v.end());

len=v.size();

for(int i=1;i<=n;i++)

p[i]=lower\_bound(v.begin(),v.end(),a[i])-v.begin()+1;

//p[i]记录a[i]离散化之后的位置

### 高精度加法

// C = A + B, A >= 0, B >= 0

vector<int> add(vector<int> &A, vector<int> &B)

{ if (A.size() < B.size()) return add(B, A);

vector<int> C;

int t = 0;

for (int i = 0; i < A.size(); i ++ )

{

t += A[i];

if (i < B.size()) t += B[i];

C.push\_back(t % 10);

t /= 10;

}

if (t) C.push\_back(t);

return C;

}

### 高精度减法

// C = A - B, 满足A >= B, A >= 0, B >= 0

vector<int> sub(vector<int> &A, vector<int> &B)

{ vector<int> C;

for (int i = 0, t = 0; i < A.size(); i ++ )

{

t = A[i] - t;

if (i < B.size()) t -= B[i];

C.push\_back((t + 10) % 10);

if (t < 0) t = 1;

else t = 0;

}

while (C.size() > 1 && C.back() == 0) C.pop\_back();

return C;

}

### 高精度乘法

// C = A \* b, A >= 0, b > 0

vector<int> mul(vector<int> &A, int b)

{ vector<int> C;

int t = 0;

for (int i = 0; i < A.size() || t; i ++ )

{

if (i < A.size()) t += A[i] \* b;

C.push\_back(t % 10);

t /= 10;

}

return C;

}

### 高精度除法

// A / b = C ... r, A >= 0, b > 0

vector<int> div(vector<int> &A, int b, int &r)

{

vector<int> C;

r = 0;

for (int i = A.size() - 1; i >= 0; i -- )

{

r = r \* 10 + A[i];

C.push\_back(r / b);

r %= b;

}

reverse(C.begin(), C.end());

while (C.size() > 1 && C.back() == 0) C.pop\_back();

return C;

}

### 高精度比较

vector<int > max\_vec(vector<int > a,vector<int > b)

{

if(a.size()>b.size() ) return a;

if(a.size()<b.size() ) return b;

if(vector<int > (a.rbegin(),a.rend())>vector< int > (b.rbegin(), b.rend()) )

return a;//当长度相同时，两个高精度由高位到低位依次进行比较

return b;

}

### 数组模拟邻接表

struct node{

int v,ne,w;

}edge[N];

int tot,head[N];

void add(int x,int y,int w)//from x to y

{

edge[++tot].v=y;

edge[tot].w=w;

edge[tot].ne=head[x];

head[x]=tot;

}

### 快读快输

inline int read()

{

char ch = getchar(); int x = 0, f = 1;

while(ch < '0' || ch > '9') {if(ch == '-') f = -1; ch = getchar();}

while('0' <= ch && ch <= '9') {x = x \* 10 + ch - '0'; ch = getchar();}

return x \* f;

}

inline int write(int x)

{

if(x<0){puctahr('-');x=~(x-1);}

int s[20],top=0;

while(x){s[++top]=x%10;x/=10;}

if(!top)s[++top]=0;

while(top)putchar(s[top--]+'0');

}

###二分查找

int bsearch\_1(int l, int r)

{

while (l < r)

{

int mid = l + r >> 1;

if (check(mid)) r = mid;

else l = mid + 1;

}

return l;

}

int bsearch\_2(int l, int r)

{

while (l < r)

{

int mid = l + r + 1 >> 1;

if (check(mid)) l = mid;

else r = mid - 1;

}

return l;

}

### 矩阵快速幂

const int N = 7;

ll n,sx,sy,dx,dy,t,mod;

struct Matrix{

int n,m;

ll v[N][N];//行、列

Matrix(int n=0,int m=0):n(n),m(m){

}

void init()

{

for(int i=0;i<n;i++)

for(int j=0;j<m;j++)

v[i][j]=0;

}

void show()

{

for(int i=0;i<n;i++)

{

for(int j=0;j<m;j++)

cout<<v[i][j]<<" ";

cout<<endl;

}

}

Matrix operator\* (const Matrix B)const

{

Matrix C(n,B.m);

C.init();

for(int i=0;i<n;i++)

for(int j=0;j<B.m;j++)

for(int k=0;k<m;k++)

C.v[i][j]=(v[i][k]\*B.v[k][j]%mod+C.v[i][j])%mod;

return C;

}

};

Matrix a,b;

void make\_a()

{

a.n=6;a.m=1;

a.init();

//赋值代码

}

void make\_b()

{

b.n=b.m=6;

b.init();

//赋值代码

}

Matrix ksm(Matrix unit,ll x,Matrix turn)

{

Matrix ret=unit;

while(x)

{

if(x&1)

ret=turn\*ret;

turn=turn\*turn;

x>>=1;

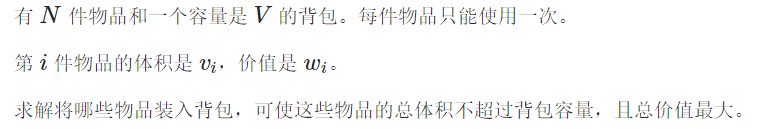
}

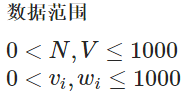
return ret;

}

#背包问题

### 01背包问题





#include<bits/stdc++.h>

using namespace std;

const int N =1e3+10;

int n,v,a[N],b[N],dp[N];

int main()

{

cin>>n>>v;

for(int i=1;i<=n;i++)

cin>>a[i]>>b[i];

for(int i=1;i<=n;i++)

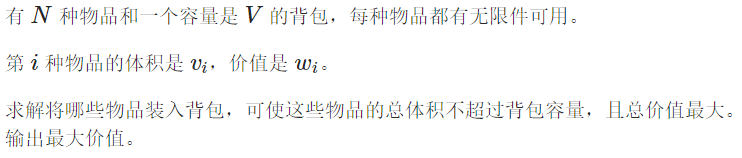
for(int j=v;j>=a[i];j--)

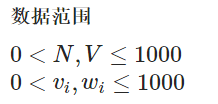
dp[j]=max(dp[j],dp[j-a[i]]+b[i]);

cout<<dp[v]<<endl;

}

### 完全背包问题





#include<bits/stdc++.h>

using namespace std;

const int N = 1e3+10;

int n,v,a[N],b[N];

int dp[N];

int main()

{

cin>>n>>v;

for(int i=1;i<=n;i++)

cin>>a[i]>>b[i];

for(int i=1;i<=n;i++)

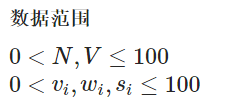
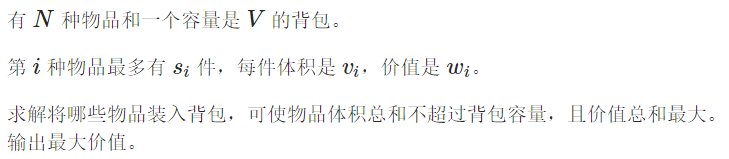
for(int j=a[i];j<=v;j++)

dp[j]=max(dp[j],dp[j-a[i]]+b[i]);

cout<<dp[v]<<endl;

}

###多重背包问题 I



#include<bits/stdc++.h>

using namespace std;

const int N = 110;

int n,v,a[N],b[N],c[N];//a:体积 b:价值 c:数量

int dp[N];

int main()

{

cin>>n>>v;

for(int i=1;i<=n;i++)

cin>>a[i]>>b[i]>>c[i];

for(int i=1;i<=n;i++)

for(int j=1;j<=c[i]&&j\*a[i]<=v;j++)

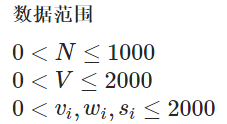
for(int k=v;k>=a[i];k--)

dp[k]=max(dp[k],dp[k-a[i]]+b[i]);

cout<<dp[v]<<endl;

}

###多重背包问题 II



#include<bits/stdc++.h>

using namespace std;

const int N = 2010;

int n,v,a,b,c;//a:体积 b:价值 c:数量

int dp[N];

struct node{

int a,b;//"二进制优化"分解之后的体积和价值

};

vector<node>s;

//复杂度n\*v\*logc

int main()

{

cin>>n>>v;

for(int i=1;i<=n;i++)

{

s.clear();

cin>>a>>b>>c;

for(int k=1;k<=c;k<<=1)

{

c-=k;

s.push\_back({k\*a,k\*b});

}

if(c)

s.push\_back({c\*a,c\*b});

for(int i=0;i<s.size();i++)

for(int j=v;j>=s[i].a;j--)

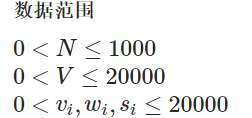
dp[j]=max(dp[j],dp[j-s[i].a]+s[i].b);

}

cout<<dp[v]<<endl;

}

###多重背包问题 III



#include<bits/stdc++.h>

using namespace std;

const int N = 2e4+10;

int n,v,a,b,c;//a:体积 b:价值 c:数量

int dp[N];

int head,tail;

struct node{

int id,val;

}qu[N];

int main()

{

cin>>n>>v;

for(int i=1;i<=n;i++)

{

cin>>a>>b>>c;

int num=min(c,v/a);

for(int j=0;j<a;j++)

{

head=1;

tail=0;

for(int k=0;k<=(v-j)/a;k++)

{

int cur=dp[k\*a+j]-k\*b;

while(head<=tail&&cur>=qu[tail].val)

tail--;

qu[++tail].id=k;

qu[tail].val=cur;

while(head<=tail&&qu[head].id<k-num)

head++;

dp[k\*a+j]=qu[head].val+k\*b;

}

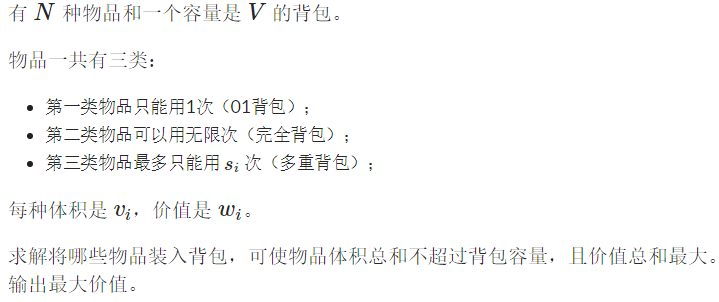
}

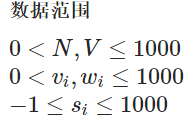
}

cout<<dp[v]<<endl;

}

###混合背包问题





#include<bits/stdc++.h>

using namespace std;

const int N = 1e4+10;

int n,v;

int a,b,c;//a:体积 b:价值 c:数量

int dp[N];

struct node{

int op,a,b;

}s[N];

int tot=0;

void add(int op,int a,int b)//存入结构体中

{

s[++tot].op=op;

s[tot].a=a;

s[tot].b=b;

}

int main()

{

cin>>n>>v;

for(int i=1;i<=n;i++)

{

cin>>a>>b>>c;

if(c==-1)

add(-1,a,b);

else if(c==0)

add(0,a,b);

else

{

for(int k=1;k<=c;k<<=1)

{

c-=k;

add(-1,a\*k,b\*k);

}

if(c)

add(-1,a\*c,b\*c);

}

}

for(int i=1;i<=tot;i++)

{

if(s[i].op==-1)

{

for(int j=v;j>=s[i].a;j--)

dp[j]=max(dp[j],dp[j-s[i].a]+s[i].b);

}

else

{

for(int j=s[i].a;j<=v;j++)

dp[j]=max(dp[j],dp[j-s[i].a]+s[i].b);

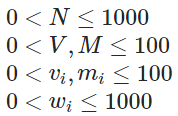
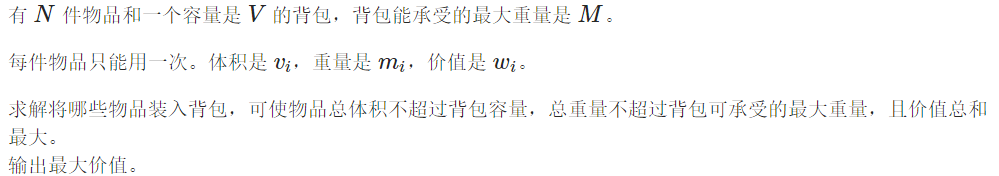
}

}

cout<<dp[v]<<endl;

}

###二维费用的背包问题



#include<bits/stdc++.h>

using namespace std;

const int N = 110;

int n,v,m;

int a,b,c;//a:体积 b:重量 c:价值

int dp[N][N];

int main()

{

cin>>n>>v>>m;

for(int i=1;i<=n;i++)

{

cin>>a>>b>>c;

for(int j=v;j>=a;j--)

for(int k=m;k>=b;k--)

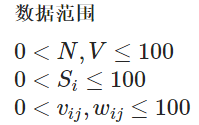
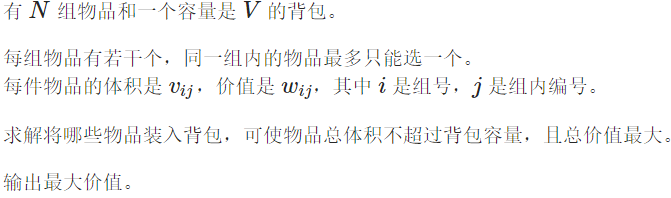
dp[j][k]=max(dp[j][k],dp[j-a][k-b]+c);

}

cout<<dp[v][m]<<endl;

}

###分组背包问题



#include<bits/stdc++.h>

using namespace std;

const int N = 110;

int n,v;

int s,a[N],b[N];//s:每组的物品数量 a:体积 b:价值

int dp[N];

int main()

{

cin>>n>>v;

for(int i=1;i<=n;i++)

{

cin>>s;

for(int j=1;j<=s;j++)

cin>>a[j]>>b[j];

for(int j=v;j>=0;j--)

{

for(int k=1;k<=s;k++)

{

if(j>=a[k])

dp[j]=max(dp[j],dp[j-a[k]]+b[k]);

}

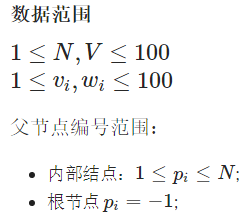
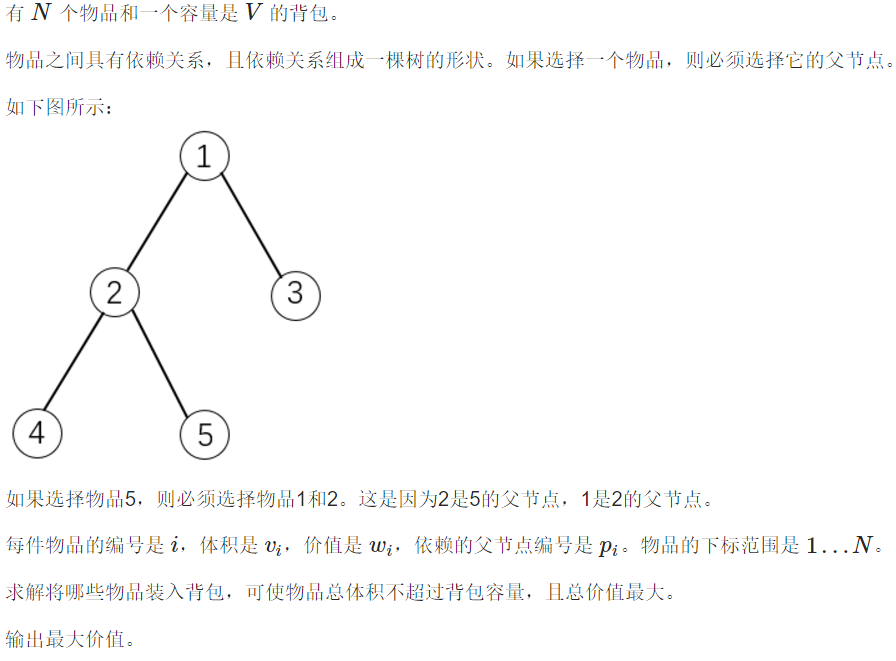
}

}

cout<<dp[v]<<endl;

}

###有依赖的背包问题



#include<bits/stdc++.h>

using namespace std;

const int N = 110;

struct node{

int v,ne;

}edge[N];

int tot,head[N];

void add(int x,int y)//from x to y

{

edge[++tot].v=y;

edge[tot].ne=head[x];

head[x]=tot;

}

int n,v;

int a[N],b[N],fa;//a:体积 b:价值

int root;

int dp[N][N];

void dfs(int now)

{

for(int i=head[now];i;i=edge[i].ne)

{

int son=edge[i].v;

dfs(son);

for(int j=v-a[now];j>=0;j--)

for(int k=0;k<=j;k++)

dp[now][j]=max(dp[now][j],dp[now][j-k]+dp[son][k]);

}

for(int i=v;i>=a[now];i--)

dp[now][i]=dp[now][i-a[now]]+b[now];

for(int i=0;i<a[now];i++)

dp[now][i]=0;

}

int main()

{

cin>>n>>v;

for(int i=1;i<=n;i++)

{

cin>>a[i]>>b[i]>>fa;

if(fa==-1)

root=i;

else

add(fa,i);

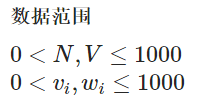
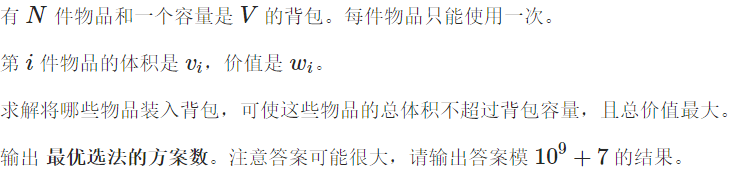
}

dfs(root);

cout<<dp[root][v]<<endl;

}

###背包问题求方案数



#include<bits/stdc++.h>

using namespace std;

typedef long long ll;

const int N = 1e3+10;

const ll mod = 1e9+7;

int n,v;

int a,b;//a:体积 b:价值

ll dp[N],cnt[N];

//dp[j]表示 背包体积总和为j时的最大价值

//cnt[j]表示 背包体积总和为j时最大价值的数目

int main()

{

cin>>n>>v;

for(int i=0;i<=v;i++)

cnt[i]=1;//赋初值

for(int i=1;i<=n;i++)

{

cin>>a>>b;

for(int i=v;i>=a;i--)

{

if(dp[i-a]+b>dp[i])

{

dp[i]=dp[i-a]+b;

cnt[i]=cnt[i-a];

}

else if(dp[i-a]+b==dp[i])

cnt[i]=(cnt[i]+cnt[i-a])%mod;

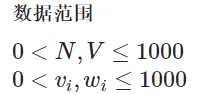
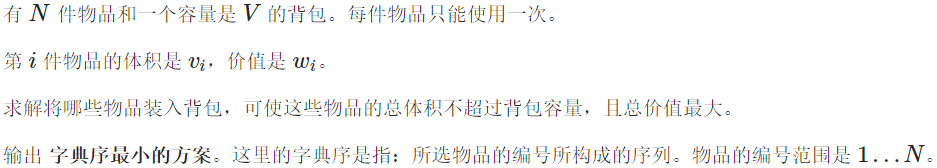
}

}

cout<<cnt[v]<<endl;

}

###背包问题求具体方案



#include<bits/stdc++.h>

using namespace std;

const int N = 1e3+10;

int n,v;

int a[N],b[N];//a:体积 b:价值

int dp[N][N];

//dp[i][j]表示:第i个物品背包总和为j的最大价值

vector<int>ans;

int main()

{

cin>>n>>v;

for(int i=1;i<=n;i++)

cin>>a[i]>>b[i];

for(int i=n;i>=1;i--)//倒序为了字典序最小

{

for(int j=0;j<=v;j++)

{

dp[i][j]=dp[i+1][j];

if(j>=a[i])

dp[i][j]=max(dp[i][j],dp[i+1][j-a[i]]+b[i]);

}

}

int res=v;

for(int i=1;i<=n;i++)

{

if(i==n&&res>=a[i])

{

ans.push\_back(i);

break;

}

if(res<=0)

break;

if(res>=a[i]&&dp[i][res]==dp[i+1][res-a[i]]+b[i])

{

ans.push\_back(i);

res-=a[i];

}

}

for(int i=0;i<ans.size();i++)

cout<<ans[i]<<" ";

cout<<endl;

}

#Trie树

### 前缀统计

求对于每个字符串，有多少前缀字符串。

#include<bits/stdc++.h>

using namespace std;

typedef long long ll;

const int N = 1e6+10;

int n,m;

string s;

int son[N][26],cnt[N],idx;

void insert()

{

int p=0;

for(int i=0;i<s.length();i++)

{

int &ne=son[p][s[i]-'a'];

if(!ne)

ne=++idx;

p=ne;

}

cnt[p]++;

}

int query()

{

int ans=0,p=0;

for(int i=0;i<s.length();i++)

{

int &ne=son[p][s[i]-'a'];

if(!ne)

break;

p=ne;

ans+=cnt[p];

}

return ans;

}

int main()

{

ios::sync\_with\_stdio(false);

cin>>n>>m;

for(int i=1;i<=n;i++)

{

cin>>s;

insert();

}

for(int i=1;i<=m;i++)

{

cin>>s;

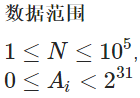
cout<<query()<<endl;

}

}

###最大异或对





#include<bits/stdc++.h>

using namespace std;

const int N = 1e5+10;

int son[32\*N][2],idx;

int n,a[N];

void insert(int num)

{

int p=0;

bitset<31>b(num);

for(int i=30;i>=0;i--)

{

int &ne=son[p][b[i]];

if(!ne)

ne=++idx;

p=ne;

}

}

int query(int num)

{

int ans=0,p=0;

bitset<31>b(num);

for(int i=30;i>=0;i--)

{

if(son[p][!b[i]])

{

ans+=1<<i;

p=son[p][!b[i]];

}

else

p=son[p][b[i]];

}

return ans;

}

int main()

{

cin>>n;

for(int i=1;i<=n;i++)

{

cin>>a[i];

insert(a[i]);

}

int ans=0;

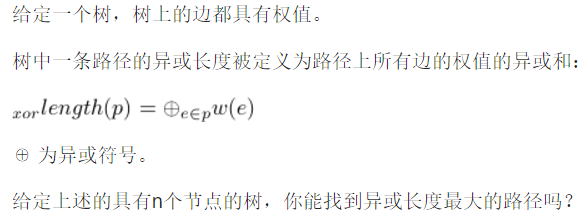
for(int i=1;i<=n;i++)

ans=max(ans,query(a[i]));

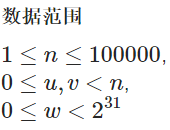
cout<<ans<<endl;

}

###最大异或值路径



求树上最大异或路径（都异或到根）



#include<bits/stdc++.h>

using namespace std;

const int N = 1e5+10;

int n,a,b,c;

int dist[N],root;

int son[32\*N][2],idx;

struct node{

int v,w,ne;

}edge[N<<1];

int head[N<<1],tot;

void add(int a,int b,int c)

{

edge[++tot].v=b;

edge[tot].w=c;

edge[tot].ne=head[a];

head[a]=tot;

}

int vis[N];

void dfs(int now)

{

vis[now]=1;

for(int i=head[now];i;i=edge[i].ne)

{

dist[edge[i].v]=dist[now]^edge[i].w;

if(!vis[edge[i].v])

dfs(edge[i].v);

}

}

void insert(int num)

{

int p=0;

bitset<31>b(num);

for(int i=30;i>=0;i--)

{

int &ne=son[p][b[i]];

if(!ne)

ne=++idx;

p=ne;

}

}

int query(int num)

{

int p=0,ans=0;

bitset<31>b(num);

for(int i=30;i>=0;i--)

{

if(son[p][b[i]^1])

{

ans+=1<<i;

p=son[p][b[i]^1];

}

else

p=son[p][b[i]];

}

return ans;

}

int main()

{

cin>>n;

for(int i=1;i<n;i++)

{

cin>>a>>b>>c;

add(a,b,c);

add(b,a,c);

}

root=0;

dfs(root);

for(int i=0;i<n;i++)

insert(dist[i]);

int res=0;

for(int i=0;i<n;i++)

res=max(res,query(dist[i]));

cout<<res<<endl;

}

# 字符串

### KMP

const int N=1e6+10;

int Next[N],b[N];

int a[N],t;

int n,m;

void GetNext()//未优化

{

int plen=m;

Next[0]=-1;

int k=-1,j=0;

while(j<plen)

{

if(k==-1||b[j]==b[k])

Next[++j]=++k;

else

k=Next[k];

}

}

void GetNext(int b[])//优化

{

int plen=m;

Next[0]=-1;

int k=-1,j=0;

while(j<plen)

{

if(k==-1||b[j]==b[k])

{

j++;k++;

if(b[j]!=b[k])

Next[j]=k;

else

Next[j]=Next[k];

}

else

k=Next[k];

}

}

**//优化之后，前缀串的i-next[i]即是循环体长度**

**//aba 循环长度为2**

void kmp(int a[],int b[])//a为文本 b为查找数组

{

int i=0,j=0,alen=n,blen=m;

while(i<alen&&j<blen)

{

if(j==-1||a[i]==b[j])

i++,j++;

else

j=Next[j];

}

//if(j==blen)

//cout<<i-j+1<<endl;

//else

//cout<<-1<<endl;

}

**//规律：如果对于next数组中的 i， 符合 i % ( i - next[i] ) == 0 && next[i] != 0 , 则说明字符串循环**

**循环节长度为: i - next[i]**

**循环次数为: i / ( i - next[i] )**

### 扩展KMP

//对于原串S的每一个后缀子串与模式串T的最长公共前缀

//next[i]表示为模式串T中以i为起点的后缀字符串和模式串T的最长公共前缀长度

//extend[i]即表示为原串S中以i为起点的后缀字符串和模式串T的最长公共前缀长度.

const int N = 1e5+10;

int Next[N],extend[N];

int getNext(string s)//预处理计算Next数组

{

int i=0,len=s.length();

Next[0]=len;

while(i+1<len&&s[i]==s[i+1])

i++;

Next[1]=i;

int po=1,j;//初始化po的位置

for(i=2;i<len;i++)

{

if(Next[i-po]+i<Next[po]+po) //第一种情况，可以直接得到next[i]的值

Next[i]=Next[i-po];

else//第二种情况，要继续匹配才能得到next[i]的值

{

j=Next[po]+po-i;

if(j<0)//如果i>po+next[po],则要从头开始匹配

j=0;

while(i+j<len&&s[j+i]==s[j])

j++;

Next[i]=j;

po=i;//更新po的位置

}

}

}

void Extend(string a,string b)//a为原串 b为模板串//计算extend数组

{

int i=0,len=a.length(),l2=b.length();

getNext(b);

while(i<l2&&i<len&&a[i]==b[i])

i++;

extend[0]=i;

int j,po=0;

for(i=1;i<len;i++)

{

if(Next[i-po]+i<extend[po]+po)//第一种情况，直接可以得到extend[i]的值

extend[i]=Next[i-po];

else //第二种情况，要继续匹配才能得到extend[i]的值

{

j=extend[po]+po-i;

if(j<0)//如果i>extend[po]+po则要从头开始匹配

j=0;

while(i+j<len&&j<l2&&a1[j+i]==a2[j])

j++;

extend[i]=j;

po=i;//更新po的位置

}

}

}

### Manacher

//最长回文串长度

const int N = 110005;

int p[N<<1];

void Manacher(string s)

{

string a="$#";//填充

int ma=0;

for(int i=0;i<s.length();i++)

{

a+=s[i];

a+='#';

p[i<<1]=0;

p[i<<1|1]=0;

}

int mi=0,r=0;//中间点、右端点

for(int i=1;i<a.length();i++)

{

if(r>i)//对于每个点，对称部分和右端距离两者取最小值

p[i]=min(p[2\*mi-i],r-i);

else

p[i]=1;

while(a[i+p[i]]==a[i-p[i]])//若还可以将区间延伸

++p[i]; //开区间

//if(a[i]=='#')

//ma=max(ma,p[i]/2\*2);

//else

//ma=max(ma,(p[i]-1)/2\*2+1);

if(r<i+p[i])

{

r=i+p[i];

mi=i;

}

}

cout<<ma<<endl;

}

#其它

### CDQ分治二维（逆序对）

int n;

int a[N],b[N];

ll ans;//ans为逆序对个数

void CDQ(int l,int r)

{

if(l==r)

return;

int mid=(l+r)>>1;

CDQ(l,mid);

CDQ(mid+1,r);

int p=l,q=mid+1,k=l;

while(p<=mid && q<=r)

{

if(a[p]<=a[q])

b[k++]=a[p++];

else

{

ans+=mid-p+1;

b[k++]=a[q++];

}

}

while(p<=mid)

b[k++]=a[p++];

while(q<=r)

b[k++]=a[q++];

for(int i=l;i<=r;++i) a[i]=b[i];

}

### CDQ分治三维（陌上花开）

#include<bits/stdc++.h>

using namespace std;

const int N = 1e5+10;

int n,k;//此题的K是a,b,c的上限

int tot;//记录总个数(除去相同的)

//求 ai<=aj且bi<=bj且ci<=cj

struct node{

int a,b,c;//a,b,c表示三个属性值

int cnt,pairs;

//cnt表示相同属性的个数，pairs表示不包含本身的个数

}arr[N],cur[N];//cur是临时值

bool cmp(const node &a,const node &b)

{

if(a.a!=b.a)

return a.a<b.a;

if(a.b!=b.b)

return a.b<b.b;

return a.c<b.c;

}

//树状数组 bgegin

int tree[N<<1];

int lowbit(int x)

{

return x&(-x);

}

void modify(int i,int x)//在第i个位置上增加x

{

while(i<=k)

{

tree[i]+=x;

i+=lowbit(i);

}

}

int query(int i) //c[1-i]之间的和

{

int s=0;

while(i>0)

{

s+=tree[i];

i-=lowbit(i);

}

return s;

}

//树状数组 end

void CDQ(int l,int r)

{

int mid=(l+r)>>1;

if(l==r)

return;

CDQ(l,mid);

CDQ(mid+1,r);

int p=l,q=mid+1,cou=l;

while(p<=mid&&q<=r)

{

if(arr[p].b<=arr[q].b)

{

modify(arr[p].c,arr[p].cnt);

cur[cou++]=arr[p++];

}

else

{

arr[q].pairs+=query(arr[q].c);

cur[cou++]=arr[q++];

}

}

while(p<=mid)

{

modify(arr[p].c,arr[p].cnt);

cur[cou++]=arr[p++];

}

while(q<=r)

{

arr[q].pairs+=query(arr[q].c);

cur[cou++]=arr[q++];

}

for(int i=l;i<=mid;i++)//清空树状数组

modify(arr[i].c,-arr[i].cnt);

for(int i=l;i<=r;i++)

arr[i]=cur[i];

}

int ans[N];

int main()

{

scanf("%d%d",&n,&k);

for(int i=1;i<=n;i++)

{

scanf("%d%d%d",&arr[i].a,&arr[i].b,&arr[i].c);

arr[i].cnt=1;

arr[i].pairs=0;

}

sort(arr+1,arr+n+1,cmp);

tot=1;

for(int i=2;i<=n;i++)

{

if(arr[i].a==arr[tot].a&&arr[i].b==arr[tot].b&&arr[i].c==arr[tot].c)

arr[tot].cnt++;

else

arr[++tot]=arr[i];

}

CDQ(1,tot);

for(int i=1;i<=tot;i++)

ans[arr[i].pairs+arr[i].cnt-1]+=arr[i].cnt;

for(int i=0;i<n;i++)

printf("%d\n",ans[i]);//满足个数为i的下标的个数

}

### CDQ分治四维

#include<bits/stdc++.h>

using namespace std;

typedef long long ll;

const int N = 5e4+10;

int n;

ll ans=0;

struct node{

int id,a,b,c;//i<j且ai<aj且bi<bj且ci<cj

int cnt,part;

//cnt表示相同属性的个数

//part记录第一维在左边还是右边

}arr[N],cur1[N],cur2[N];

int pairs[N];

//pairs表示不包含本身的个数

//树状数组 bgegin

int tree[N];

int lowbit(int x)

{

return x&(-x);

}

void modify(int i,int x)//在第i个位置上增加x

{

while(i<=n)

{

tree[i]+=x;

i+=lowbit(i);

}

}

int query(int i) //c[1-i]之间的和

{

int s=0;

while(i>0)

{

s+=tree[i];

i-=lowbit(i);

}

return s;

}

void myclear(int i)

{

while(i<=n)

{

if(tree[i]==0)

break;

tree[i]=0;

i+=lowbit(i);

}

}

//树状数组 end

void cdq(int l,int r)

{

int mid=(l+r)>>1;

if(l>=r)

return;

cdq(l,mid);

cdq(mid+1,r);

int p=l,q=mid+1,cou=l;

//这里的cur1数组已经是对b有序的一个数组,

//part用于判断下标,part=1代表在(l,mid),part=0代表在(mid+1,r)区间

while(p<=mid&&q<=r)

{

if(cur1[p].b<cur1[q].b)

{

if(cur1[p].part)

modify(cur1[p].c,1);

cur2[cou++]=cur1[p++];

}

else

{

if(!cur1[q].part)

pairs[cur1[q].id]+=query(cur1[q].c);

cur2[cou++]=cur1[q++];

}

}

while(p<=mid)

cur2[cou++]=cur1[p++];

while(q<=r)

{

if(!cur1[q].part)

pairs[cur1[q].id]+=query(cur1[q].c);

cur2[cou++]=cur1[q++];

}

for(int i=l;i<=r;i++)

{

myclear(cur1[i].c);

cur1[i]=cur2[i];

}

}

void CDQ(int l,int r)

{

int mid=(l+r)>>1;

if(l>=r)

return;

CDQ(l,mid);

CDQ(mid+1,r);

int p=l,q=mid+1,cou=l;

while(p<=mid&&q<=r)

{

if(arr[p].a<arr[q].a)

{

arr[p].part=1;

cur1[cou++]=arr[p++];

}

else

{

arr[q].part=0;

cur1[cou++]=arr[q++];

}

}

while(p<=mid)

{

arr[p].part=1;

cur1[cou++]=arr[p++];

}

while(q<=r)

{

arr[q].part=0;

cur1[cou++]=arr[q++];

}

for(int i=l;i<=r;i++)

arr[i]=cur1[i];

cdq(l,r);

}

int main()

{

scanf("%d",&n);

for(int i=1;i<=n;i++)

{

scanf("%d",&arr[i].a);

arr[i].id=i;

}

for(int i=1;i<=n;i++)

scanf("%d",&arr[i].b);

for(int i=1;i<=n;i++)

scanf("%d",&arr[i].c);

//第一维为下标，所以此处不用排序

CDQ(1,n);

for(int i=1;i<=n;i++)

ans+=(ll)pairs[i];

printf("%lld\n",ans);

}

### 树状数组

//树状数组 begin

int tree[N];

int lowbit(int x)

{

return x&(-x);

}

void modify(int i,int x)//在第i个位置上增加x

{

while(i<=n)

{

tree[i]+=x;

i+=lowbit(i);

}

}

int query(int i) //c[1-i]之间的和

{

int s=0;

while(i>0)

{

s+=tree[i];

i-=lowbit(i);

}

return s;

}

void myclear(int i)

{

while(i<=n)

{

if(tree[i]==0)

break;

tree[i]=0;

i+=lowbit(i);

}

}

//树状数组 end

### 莫队

**### 莫队板子题**

#include<bits/stdc++.h>

using namespace std;

typedef long long ll;

const int N = 2e5+10;

int n,q,a[N];

int block[N],blocken;

int num[1000010];

int ans[N],cnt;

struct node{

int l,r,id;

}arr[N];

bool cmp(const node &a,const node &b)

{

return (block[a.l]^block[b.l])?block[a.l]<block[b.l]:((block[a.l]&1)?a.r<b.r:a.r>b.r);

}

void add(int pos)

{

if(!num[a[pos]])

cnt++;

num[a[pos]]++;

}

void del(int pos)

{

num[a[pos]]--;

if(!num[a[pos]])

cnt--;

}

void MO()

{

int l=arr[1].l,r=arr[1].l-1;

for(int i=1;i<=q;i++)

{

while(arr[i].l>l)

del(l++);

while(arr[i].l<l)

add(--l);

while(arr[i].r>r)

add(++r);

while(arr[i].r<r)

del(r--);

ans[arr[i].id]=cnt;

}

}

int main()

{

scanf("%d",&n);

for(int i=1;i<=n;i++)

scanf("%d",&a[i]);

scanf("%d",&q);

for(int i=1;i<=q;i++)

{

scanf("%d%d",&arr[i].l,&arr[i].r);

arr[i].id=i;

}

blocken=sqrt(n);

for(int i=1;i<=n;i++)

block[i]=i/blocken;

sort(arr+1,arr+q+1,cmp);

MO();

for(int i=1;i<=q;i++)

printf("%d\n",ans[i]);

}

**### 带修莫队**

/\*

题意：

1、 Q L R代表询问你从第L支画笔到第R支画笔中共有几种不同颜色的画笔。

2、 R P Col 把第P支画笔替换为颜色Col

\*/

#include<bits/stdc++.h>

using namespace std;

typedef long long ll;

const int N = 1e6+10;

int n,m,col[N],last[N];

int blocken,block[N];

int num[N],cnt,ans[N];

struct node{

int l,r,t,id;

}arr[N];

struct query{

int pos,pre,now;

}q[N];

int cnt1,cnt2;

char c[5];

bool cmp(const node& a, const node& b)

{

return block[a.l]^block[b.l]?block[a.l]<block[b.l]:

block[a.r]^block[b.r]?block[a.r]<block[b.r]:

a.t<b.t;

}

void add(int pos)

{

if(!num[col[pos]]++)

cnt++;

}

void del(int pos)

{

if(!--num[col[pos]])

cnt--;

}

void MO()

{

int l=arr[1].l,r=arr[1].l-1,t=0;

for(int i=1;i<=cnt1;i++)

{

while(arr[i].t>t)

{

t++;

if(q[t].pos>=l&&q[t].pos<=r)

{

if(!--num[col[q[t].pos]])

cnt--;

if(!num[q[t].now]++)

cnt++;

}

col[q[t].pos]=q[t].now;

}

while(arr[i].t<t)

{

if(q[t].pos>=l&&q[t].pos<=r)

{

if(!--num[col[q[t].pos]])

cnt--;

if(!num[q[t].pre]++)

cnt++;

}

col[q[t].pos]=q[t].pre;

t--;

}

while(arr[i].l>l)

del(l++);

while(arr[i].l<l)

add(--l);

while(arr[i].r>r)

add(++r);

while(arr[i].r<r)

del(r--);

ans[arr[i].id]=cnt;

}

}

int main()

{

scanf("%d%d",&n,&m);

for(int i=1;i<=n;i++)

{

scanf("%d",&col[i]);

last[i]=col[i];

}

for(int i=1;i<=m;i++)

{

scanf("%s",c);

if(c[0]=='Q')

{

cnt1++;

scanf("%d%d",&arr[cnt1].l,&arr[cnt1].r);

arr[cnt1].id=cnt1;

arr[cnt1].t=cnt2;

}

else

{

int id,v;

scanf("%d%d",&id,&v);

q[++cnt2].pos=id;

q[cnt2].now=v;

q[cnt2].pre=last[id];

last[id]=v;

}

}

blocken=pow(n,2.0/3);

for(int i=1;i<=n;i++)

block[i]=i/blocken;

sort(arr+1,arr+cnt1+1,cmp);

MO();

for(int i=1;i<=cnt1;i++)

printf("%d\n",ans[i]);

}

###线段树

//线段树begin

struct seg{

int l,r,val;

}tree[N<<2];

int lazy[N<<2];

inline void build(int k,int l,int r)//k为当前值 建树

{

tree[k].l=l;tree[k].r=r;

if(l==r)

{

tree[k].val=wt[l];

return;

}

int mid=(l+r)>>1;

build(k<<1,l,mid);

build(k<<1|1,mid+1,r);

tree[k].val=tree[k<<1].val+tree[k<<1|1].val;

}

inline void pushdown(int pos)//向下传递

{

if(lazy[pos]!=-1)

{

lazy[pos<<1]=lazy[pos];

lazy[pos<<1|1]=lazy[pos];

tree[pos<<1].val=(tree[pos<<1].r-tree[pos<<1].l+1)\*lazy[pos];

tree[pos<<1|1].val=(tree[pos<<1|1].r-tree[pos<<1|1].l+1)\*lazy[pos];

lazy[pos]=-1;

}

}

inline void modify\_val(int pos,int l,int r,ll val)

{

if(tree[pos].l>=l&&tree[pos].r<=r)

{

lazy[pos]=val;

tree[pos].val=(tree[pos].r-tree[pos].l+1)\*val;

// cout<<l<<" "<<r<<" "<<val<<endl;

return;

}

pushdown(pos);

int mid=(tree[pos].l+tree[pos].r)/2;

if(l<=mid)

modify\_val(pos<<1,l,r,val);

if(mid<r)

modify\_val(pos<<1|1,l,r,val);

tree[pos].val=tree[pos<<1].val+tree[pos<<1|1].val;

}

inline int query\_add(int pos,int l,int r)

{

if(tree[pos].l>=l&&tree[pos].r<=r)

{

// cout<<l<<" "<<r<<" "<<tree[pos].val<<endl;

return tree[pos].val;

}

pushdown(pos);

int ans=0;

int mid=(tree[pos].l+tree[pos].r)/2;

if(mid>=l)

ans=ans+query\_add(pos<<1,l,r);

if(mid<r)

ans=ans+query\_add(pos<<1|1,l,r);

return ans;

}

//线段树end

//线段树上交换维护的最大最小值操作

struct seg{

int l,r,ma,mi,val;

}tree[N<<2];

int lazy[N<<2];

void build(int k,int l,int r)//k为当前值 建树

{

tree[k].l=l;tree[k].r=r;

if(l==r)

{

if(l==1)

{

tree[k].ma=-INF;

tree[k].mi=INF;

tree[k].val=0;

return;

}

tree[k].ma=wt[l];

tree[k].mi=wt[l];

tree[k].val=wt[l];

return;

}

int mid=(l+r)>>1;

build(k<<1,l,mid);

build(k<<1|1,mid+1,r);

tree[k].val=tree[k<<1].val+tree[k<<1|1].val;

tree[k].ma=max(tree[k<<1].ma,tree[k<<1|1].ma);

tree[k].mi=min(tree[k<<1].mi,tree[k<<1|1].mi);

}

void pushdown(int pos)

{

if(lazy[pos]!=1)

{

tree[pos<<1].mi=-tree[pos<<1].mi;

tree[pos<<1].ma=-tree[pos<<1].ma;

swap(tree[pos<<1].mi,tree[pos<<1].ma);

tree[pos<<1|1].mi=-tree[pos<<1|1].mi;

tree[pos<<1|1].ma=-tree[pos<<1|1].ma;

swap(tree[pos<<1|1].mi,tree[pos<<1|1].ma);

tree[pos<<1].val=-tree[pos<<1].val;

tree[pos<<1|1].val=-tree[pos<<1|1].val;

lazy[pos<<1]=-lazy[pos<<1];

lazy[pos<<1|1]=-lazy[pos<<1|1];

lazy[pos]=1;

}

}

void modify\_change(int pos,int l,int val)//单点修改

{

if(tree[pos].l==tree[pos].r)

{

tree[pos].val=val;

tree[pos].ma=val;

tree[pos].mi=val;

return;

}

pushdown(pos);

int mid=(tree[pos].l+tree[pos].r)/2;

if(l<=mid)

modify\_change(pos<<1,l,val);

else

modify\_change(pos<<1|1,l,val);

tree[pos].val=tree[pos<<1].val+tree[pos<<1|1].val;

tree[pos].ma=max(tree[pos<<1].ma,tree[pos<<1|1].ma);

tree[pos].mi=min(tree[pos<<1].mi,tree[pos<<1|1].mi);

}

void modify\_reverse(int pos,int l,int r)//将区间内的值变为相反数 即乘-1

{

if(tree[pos].l>=l&&tree[pos].r<=r)

{

tree[pos].ma=-tree[pos].ma;

tree[pos].mi=-tree[pos].mi;

tree[pos].val=-tree[pos].val;

swap(tree[pos].ma,tree[pos].mi);

lazy[pos]=-lazy[pos];

return;

}

pushdown(pos);

int mid=(tree[pos].l+tree[pos].r)/2;

if(l<=mid)

modify\_reverse(pos<<1,l,r);

if(mid<r)

modify\_reverse(pos<<1|1,l,r);

tree[pos].val=tree[pos<<1].val+tree[pos<<1|1].val;

tree[pos].ma=max(tree[pos<<1].ma,tree[pos<<1|1].ma);

tree[pos].mi=min(tree[pos<<1].mi,tree[pos<<1|1].mi);

}

int query\_add(int pos,int l,int r)

{

if(tree[pos].l>=l&&tree[pos].r<=r)

{

// cout<<l<<" "<<r<<" "<<tree[pos].val<<endl;

return tree[pos].val;

}

pushdown(pos);

int ans=0;

int mid=(tree[pos].l+tree[pos].r)/2;

if(mid>=l)

ans=ans+query\_add(pos<<1,l,r);

if(mid<r)

ans=ans+query\_add(pos<<1|1,l,r);

return ans;

}

int query\_max(int pos,int l,int r)

{

if(tree[pos].l>=l&&tree[pos].r<=r)

return tree[pos].ma;

pushdown(pos);

int ans=-INF;

int mid=(tree[pos].l+tree[pos].r)/2;

if(mid>=l)

ans=max(ans,query\_max(pos<<1,l,r));

if(mid<r)

ans=max(ans,query\_max(pos<<1|1,l,r));

return ans;

}

int query\_min(int pos,int l,int r)

{

if(tree[pos].l>=l&&tree[pos].r<=r)

return tree[pos].mi;

pushdown(pos);

int ans=INF;

int mid=(tree[pos].l+tree[pos].r)/2;

if(mid>=l)

ans=min(ans,query\_min(pos<<1,l,r));

if(mid<r)

ans=min(ans,query\_min(pos<<1|1,l,r));

return ans;

}

### 最小生成树

//最小生成树

//prim

//适用于稠密图，时间复杂度 O(n^2)

// st[i] 表示点i是否在当前生成树集合中

// dist[i] 表示点i到当前集合的最短边的长度

// g[i][j] 表示点i和点j之间边的长度

// 返回值：最小生成树中所有边的总长度

int Prim()

{

int res = 0;

for (int i = 1; i <= n; i ++ )

{

dist[i] = INF;

st[i] = false;

}

dist[1] = 0;

for (int i = 1; i <= n; i ++ )

{

int id = -1, min\_dist = INF;

// 寻找最短边

for (int j = 1; j <= n; j ++ )

if (!st[j] && dist[j] < min\_dist)

{

id = j;

min\_dist = dist[j];

}

st[id] = true;

res += dist[id];

// 用新加入的点更新其余点到生成树的最短边

for (int j = 1; j <= n; j ++ )

if (!st[j])

dist[j] = min(dist[j], g[id][j]);

}

return res;

}

//Kruskal

//适用于稀疏图，时间复杂度 O(mlogm)

// 边的信息

struct Edge

{

int a, b, v;

bool operator< (const Edge &W) const

{

return v < W.v;

}

};

// 并查集——寻找当前集合的代表元素

int find(int x)

{

if (father[x] != x) father[x] = find(father[x]);

return father[x];

}

// 所有边存储在 Edge edges[M];

// 函数返回最小生成树中所有边的总长度

int Kruskal()

{

int res = 0;

// 初始化并查集代表元素

for (int i = 1; i <= n; i ++ ) father[i] = i;

sort(edge, edge + m);

for (int i = 0; i < m; i ++ )

{

int a = edge[i].a, b = edge[i].b;

if (find(a) != find(b))

{

res += edge[i].v;

father[find(a)] = find(b);

}

}

return res;

}

### floyd

ll edge[M][M],dis[M][M];

ll floyd()//求最小路径环

{

for(int i=1;i<=tot;i++)

for(int j=1;j<=tot;j++)

edge[i][j]=dis[i][j]=INF;

for(int i=1;i<=tot;i++)

{

for(int j=i+1;j<=tot;j++)

{

if(a[i]&a[j])

{

edge[i][j]=edge[j][i]=1;

dis[i][j]=dis[j][i]=1;

}

}

}

ll ans=INF;

for(int k=1;k<=tot;k++)

{

for(int i=1;i<k;i++)

for(int j=i+1;j<k;j++)

ans=min(ans,dis[i][j]+edge[i][k]+edge[k][j]);

for(int i=1;i<=tot;i++)

for(int j=1;j<=tot;j++)

dis[i][j]=min(dis[i][j],dis[i][k]+dis[k][j]);

}

if(ans==INF)

return -1;

else

return ans;

}

### 单调队列

//直方图（栈）

ll n,h[N],qu[N],ma,sum;

ll ld[N],rd[N];

void solve(ll ld[])

{

ll l=0,r=0;

h[0]=-1;

for(int i=1;i<=n;i++)

{

while(r>l&&h[qu[r]]>=h[i])//单调递增

r--;

ld[i]=qu[r]+1;

qu[++r]=i;

}

}

int main()

{

while(~scanf("%lld",&n))

{

if(n==0)

break;

ma=0;

for(int i=1;i<=n;i++)

scanf("%lld",&h[i]);

solve(ld);

reverse(h+1,h+n+1);

solve(rd);

reverse(h+1,h+n+1);

// for(int i=1;i<=n;i++)

// cout<<ld[i]<<" "<<rd[i]<<endl;

for(int i=1;i<=n;i++)

ma=max(ma,h[i]\*(n+2-rd[n+1-i]-ld[i]));

printf("%lld\n",ma);

}

}

//双端队列

void solve()

{

ll l=1,r=0,ans1=0,ans2=0;

for(int i=1;i<m;i++)

{

while(r>=l&&a[i]>=a[qu[r]])

r--;

qu[++r]=i;

}

for(int i=m;i<=n;i++)

{

while(i-qu[l]+1>m&&l<=r)

l++;

while(r>=l&&a[i]>=a[qu[r]])

r--;

qu[++r]=i;

}

}

//最大值和最小值之差在m-k之间的最长子序列

l1=l2=1;

r1=r2=0;

ans=now=0;

for(int i=1;i<=n;i++)

{

while(l1<=r1&&a[qu1[r1]]<a[i])

r1--;

qu1[++r1]=i;

while(l2<=r2&&a[qu2[r2]]>a[i])

r2--;

qu2[++r2]=i;

while(r1>=l1&&r2>=l2&&a[qu1[l1]]-a[qu2[l2]]>k)

{

now=min(qu1[l1],qu2[l2]);

if(now==qu1[l1])

l1++;

if(now==qu2[l2])

l2++;

}

if(r1>=l1&&r2>=l2&&a[qu1[l1]]-a[qu2[l2]]>=m)

ans=max(ans,i-now);

}

### 第K小完全图

#include<bits/stdc++.h>

using namespace std;

typedef long long ll;

const int N = 105;

int n,k;

bool ma[N][N];

ll u[N];

struct node{

ll weight;

int z;

vector<int>dis;

bool operator<(const node &x)const

{

return weight>x.weight;

}

}a[N],cur;

priority\_queue<node>q;

int main()

{

ios::sync\_with\_stdio(false);

cin>>n>>k;

for(int i=1;i<=n;i++)

cin>>u[i];

for(int i=1;i<=n;i++)

{

string s;

cin>>s;

for(int j=0;j<s.length();j++)

ma[i][j+1]=(s[j]=='1'?1:0);

}

k--;//去掉0；

if(k==0)

{

cout<<0<<endl;

return 0;

}

for(int i=1;i<=n;i++)

{

a[i].weight=u[i];

a[i].z=0;

a[i].dis.push\_back(i);

q.push(a[i]);

}

while(!q.empty())

{

cur=q.top();

k--;

q.pop();

if(k==0)

{

cout<<cur.weight<<endl;

return 0;

}

int now=cur.dis[cur.z]+1;

for(int i=now;i<=n;i++)

{

bool flag=1;

for(int j=0;j<=cur.z;j++)

{

if(ma[cur.dis[j]][i]==0)

{

flag=0;

break;

}

}

if(flag)

{

cur.weight+=u[i];

cur.z++;

cur.dis.push\_back(i);

q.push(cur);

cur.weight-=u[i];

cur.z--;

cur.dis.pop\_back();

}

}

}

cout<<-1<<endl;

return 0;

}